Towards Large-Scale and Long-Term Neural Map Representations

Ming-Fang (Allie) Chang

CMU-RI-TR-23-66

September 25, 2023

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

Abstract

We address the problem of large-scale and long-term neural map representations. Maps provide valuable information for modern robotic applications such as autonomous driving and AR/VR. In this thesis, we explored two important perspectives of map design: size and richness. First, we look into the map compression problem for image-to-LiDAR map, LiDAR-to-LiDAR map, and image-to-SfM map registration. For image-to-LiDAR map registration, we proposed a learning-based technique to precompute and compress a voxelized LiDAR map before performing image registration. For LiDAR-to-LiDAR map registration, we performed map compression benchmarks for existing deep learning based and traditional methods. For image-to-SfM map registration, we proposed selecting important keypoints from a SfM map through a heterogeneous graph neural network. The outcomes of all the three works lead to significant reduction of map size with offline preprocessing, and thus offloads the data burden of online image registration.

Second, inspired by the promising results of recent NeRF works, we developed a LiDAR-assisted NeRF system that encodes the rich appearance and geometry details of an outdoor environment into point-based neural representation and performs novel view synthesis. Unlike most of the previous NeRF works that focus on indoor or small scenes, our system is designed for more challenging canonical autonomous driving datasets such as Argoverse 2, which has scarcer training views and larger scene complexity. We use a point-based NeRF framework with a conditional GAN, and successfully outperformed state-of-the-art outdoor NeRF baselines. In addition, we explored several applications for outdoor NeRFs, including data augmentation, object detection, and seasonal view synthesis. Our experiments show the foreseeable potential of applying neural representation for more practical outdoor applications in the future.

Acknowledgments

I wish to express my sincere gratitude to the esteemed members of my thesis committee: Simon, Michael, Matt, and Ian, for their invaluable presence on my committee and for their insightful contributions. I extend my heartfelt appreciation to my PhD advisors, Simon and Michael, for their unwavering inspiration, understanding, and support throughout the highs and lows of my doctoral journey. The wisdom imparted by you will undoubtedly shape my future works.

I would like to acknowledge the collaboration of all individuals who have been part of this journey. It is my honor to have been co-advised and to have participated in two laboratories. My gratitude goes to my great labmates from the CI2CV lab, including Kit, Ash, Chen-Hsuan, Chaoyang, Nate, and all other members. Their consistent support from the beginning of my CMU days is truly appreciated. I must also thank my fellow labmates in RPL, namely Josh, Ming, Eric, Paloma, Wei, Suddhu, Monty, Akshay, Akash, and others, for their assistance and enriching conversations during our collective meetings and projects. Their presence has undoubtedly illuminated my PhD life.

Furthermore, I am grateful for the Argoverse 1 team, including James, Patsorn, John, and Jagjeet, as well as all the former Argo AI collaborators. Collaborating with this exceptional team at the start of my PhD journey was indispensable. I also deeply appreciate the people I met in Meta: my internship mentor Yipu, Rajvi, and Jakob. Not only that I got the chance to interact with these excellent people through internships, but also that the insights gained and experiences shared have significantly enriched my following academic path.

Finally, my sincere gratitude goes to my parents, husband Bo-Yi, feline friend Oreo, and newborn daughter Rae, for their unconditional companion, love and support. I also thank my dearest volleyball friends: Shoou-I, Yusan, Ming, Mini, Gary, and others, where hours of play and Fukutea conversations remain among my fondest CMU memories. This moment signifies not an end, but a new beginning. I look forward to what we can accomplish in the future with great appreciation to the past.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

Maps, as our prior understanding of the environment, play essential roles for many modern robotic applications. A map is a task-oriented representation that provides necessary information to facilitate the targeted task. Depending on the system design and application, a map can carry geometric structures, appearance details, semantic labels, landmarks, etc. For example, autonomous vehicles rely on the sign locations and road lines in High-Definition (HD) maps to navigate through cities [24]. An AR/VR device relies on the 3D geometry in a map to localize and enable the user to interact with the environment [77]. Without the maps, a robot would have to rely on limited instantaneous sensor inputs to operate and is prone to failure, especially in large-scale and long-term applications due to error accumulation. The design of maps, however, is a non-trivial art of balance between storage and richness. The richer information a map carries, the more tasks it might support, but the more storage it would require.

This thesis is a combination of our works on map design through my PhD, where we looked into different sensor modalities and representations. It is structured as follows: first, we introduce our efforts for developing compressive map representations for image-to-LiDAR registration [26] (Chap. 3) and LiDAR-to-LiDAR map registration [25] (Chap. 4). Second, we describe a method for reducing the map size (i.e. the number of feature points) for long-term image-to-SfM map registration [27] (Chap. 5). Finally, we present a LiDAR-assisted Neural Radiance Field system that encodes rich information for outdoor environments (Chap. 6) for novel view synthesis. Summaries of each work are listed below:

1.1 Image-to-LiDAR map registration

This work addresses the problem of image registration to a compressed 3D map. While this is most often performed by comparing LiDAR scans to the point cloud based map [166], it depends on an expensive LiDAR sensor at run time and the large point cloud based map creates overhead in data storage and transmission. Previously, efforts have been underway to replace the expensive LiDAR sensor with cheaper cameras and perform 2D-3D localization [19, 22, 67, 89].

In contrast to the previous work that learns relative pose by comparing projected depth and camera images [22], we propose HyperMap [26], a paradigm shift from online depth map feature extraction to offline 3D map feature computation for the 2D-3D camera registration task through end-to-end training. In the proposed pipeline, we first perform offline 3D sparse convolution to extract and compress the voxelwise hypercolumn features for the whole map. Then at run-time, we project and decode the compressed map features to the rough initial camera pose to form a virtual feature image. A Convolutional Neural Network (CNN) is then used to predict the relative pose between the camera image and the virtual feature image. In addition, we propose an efficient occlusion handling layer, specifically designed for large point clouds, to remove occluded points in projection. Our experiments on synthetic and real datasets show that, by moving the feature computation load offline and compressing, we reduced map size by %87 – 94.

1.2 LiDAR-to-LiDAR map registration

Modern autonomous vehicles utilize pre-built HD (High-Definition) maps to perform sensor-to-map registration, which recovers pose estimation failures and reduces drift in a large-scale environment. However, sensor-to-map registration is usually realized by registering the sensor to a dense 3D model that occupies massive storage space in the HD map and requires much data processing overhead. Although smaller 3D models are preferable, the optimal compressive map format for preservation of the best registration performance remains unclear.

Here we propose a novel and challenging benchmark to evaluate existing LiDAR-to-map registration methods from three perspectives: map compressibility, robustness, and precision [25]. We compared various map formats, including raw points, hierarchical GMMs, and feature points, and show their performance trade-offs between compressibility and robustnesson real-world LiDAR datasets: KITTI Odometry Dataset [51] and Argoverse Tracking Dataset [24]. Our benchmark reveals that state-of-the-art deep feature point based methods outperform traditional methods significantly when the map size budget is high. However, when map size budget is low, deep methods are outperformed by the methods using simpler models in Argoverse Tracking Dataset due to poor spatial coverage. In addition, we observe that TEASER++ [156] significantly outperforms RANSAC for the feature point methods. Our analysis provides a valuable reference for the community to design budgeted real-world systems and find potential research opportunities. We released the benchmark for public use.

1.3 Image-to-SfM map registration

In this work, we address the problem of map sparsification for long-term visual localization [27]. For map sparsification, a commonly employed assumption is that the pre-build map and the later captured localization query are consistent. However, this assumption can be easily violated in the dynamic world. Additionally, the map size grows as new data accumulate through time, causing large data overhead in the long term. In this work, we aim to overcome the environmental changes and reduce the map size at the same time by selecting points that are valuable to future localization.

Inspired by the recent progress in Graph Neural Network (GNN), we propose the first work that models SfM maps as heterogeneous graphs and predicts 3D point importance scores with a GNN, which enables us to directly exploit the rich information in the SfM map graph. Two novel supervisions are proposed: 1) a data-fitting term for selecting valuable points to future localization based on training queries; 2) a K-Cover term for selecting sparse points with full-map coverage. The experiments show that our method selected map points on stable and widely visible structures and outperformed baselines in localization performance.

1.4 LiDAR-assisted Neural Radiance Field

Existing NeRF methods usually require densely-sampled source views and do not perform well with the open source camera-LiDAR datasets. In this work, we demonstrate that such datasets [17, 150] can be used to construct high quality neural renderings.

Our design leverages 1) LiDAR sensors for strong 3D geometry priors that significantly improve the ray sampling locality, and 2) Conditional Adversarial Networks (cGANs) [65] to recover image details since aggregating embeddings from imperfect LiDAR maps causes artifacts. Our experiments show that while NeRF baselines produce either noisy or blurry results on Argoverse 2 [150], our system not only outperforms baselines in image quality metrics under both clean and noisy conditions, but also obtains closer Detectron2 [153] results to the ground truth images. Furthermore, this system can be used in data augmentation for training a pose regression network [12] and multi-season view synthesis. This work serves as a new LiDAR-based NeRF baseline that pushes this research direction forward.

In summary, we looked into multiple sensors (camera, LiDAR), and map representations (voxel, point cloud, neural radiance field), and also proposed methods to improve the performance of specific tasks (map compression, novel view synthesis) in each work. Importantly, I would like to point out that this document collects contributions from my collaborators in the corresponding projects. By sharing our experience, we hope this document can benefit future researchers–in robotics or computer vision–in achieving better system design for map related applications.

Chapter 2

Related Works

In this chapter, we list the related works to the methods described in the following chapters. The scope of our related works is broad, so we categorized them into three parts: sensor registration (Sec. 2.1), map compression (Sec. 2.2), and neural radiance field (Sec. 2.3).

2.1 Sensor Registration

Sensor registration refers to the process of localizing a sensor measurement on a pre-built map in this work. It can occur between different modality combinations for the sensor and the map (e.g. images, LiDAR), but the spirit is all about matching the common parts in the sensor measurement and the map that correspond to the same geographical location. We focus on local registration, or local localization, approaches that refine a rough initial pose estimate by matching online sensor measurement to a map. In contrast to local registration, there are also global registration methods that do not require an initial pose but are usually less accurate [21, 50, 66, 117, 130].

2.1.1 Image to LiDAR map

LiDAR sensors are more accurate than cameras for measuring scene geometry but are also much more expensive. An economical compromise is to use the expensive LiDAR in map building, and than perform localization with only cameras. In this way, only the devices used in mapping need to be equiped with LiDAR sensors.

Explicit Methods

Leveraging classical visual odometry methods, Caselitz at el. [19] used Structure-from-Motion (SfM) to reconstruct sparse point clouds from video sequences and then performed ICP to register the SfM point cloud to the LiDAR map, which requires robust feature points and video sequences, not a single image. Kim et al. proposed to register a stereo camera to a LiDAR map using the image feature correspondences in stereo depth and projective LiDAR depth [67]. Mastin et al. proposed to use mutual information to register an aerial image to LiDAR images [89].

In autonomous driving applications, the ground plane and road markings can be especially useful. Lu et al. [82] used the chamfer distance to align the detected road markings to a sparse 3D map. Wolcott and Eustice [151] proposed to use reflectance information derived from the LiDAR ground map, containing mostly road marking information, to solve the local registration problem. Their method generated synthetic projective reflectance images and refined the initial pose by maximizing the mutual information score to align the synthetic reflectance images with a monocular camera onboard the vehicle. While the ground plane provides a distinctive set of features for alignment, methods that depend on it fail when large portions of the road are occluded or differ from the pre-built map, such as in the presence of snow or after construction [152]. This dependence on the ground-plane can be overcome by taking into account the 3D-volumetric information. In [152], Wolcott and Eustice proposed the use of Gaussian-mixture-models (GMMs) to summarize map height and reflectivity for efficient LiDAR-based localization. However, this method also depends on having a LiDAR sensor on-board the vehicle at localization time. Some works used 2D-3D line correspondences for registration, which only works when line features exist [33, 162].

Deep Neural Networks

In 2019, CMRNet [20] and CMRNet++ [22] leveraged a CNN to solve the local 2D-3D registration problem in a way that both takes into account 3D-structure information and only requires a monocular camera at localization time. It adopted a correlation filter, which is often used in the optical flow networks [129], to regress the relative 6-DoF pose between a virtual LiDAR depth image and an RGB camera image. The experiments showed that CMRNet reached centimeter-level translation error in an unseen environment. EnforceNet [30] also used a CNN to regress pose between projected depth and RGB images. In 2021, we proposed our method, HyperMap [26], that first computes 3D on-map features, projecting the computed features onto a virtual image plane, and than regresses the relative pose with a CNN. We named this strategy "*late projection*" because the projection step happens after 3D feature computation. The late projection strategy can greatly reduce the map size by preprocessing and storing the map as compressed features. To the extent of our knowledge, CMRNet was the only existing method that, like our HyperMap, was not trained in testing environment, making it more generalizable to maps other than the ones on which it was trained. Thus we pick CMRNet as our baseline. Both CMRNet and EnforceNet perform all feature computation after projection, we refer to this design as "*early projection*".

More recently, to solve this cross-modality matching problem, several learning-based methods that support end-to-end training are developed. One can use a rgb-to-depth network that transforms the image to depth domain for matching [74], or an image-to-LiDAR correspondence flow estimation network [29] whose output can be used by PnP to compute relative poses. In addition to the CNN with a correlation module used by [20, 22, 26], a transformer can be added to regress the relative pose from CNN outputs and position embeddings [92].

Occlusion Handling. Occlusion handling is an important module in the differential renderers used in 3D shape learning to project 3D information to 2D. It is needed for end-to-end learning when the training gradient propagates from 2D to 3D embeddings on the LiDAR map. Lin et al. proposed using upsampling and max-pooling process to build a pseudo-renderer [78]. In [137] and [64], the authors used a differentiable ray tracing method with probabilistic voxel occupancy for occlusion reasoning. Sitzmann et al. proposed an occlusion-aware projection that first transformed the voxelized feature representation to the canonical view grid and then used a network to predict the per-pixel visibility [127].

In our case, the outdoor LiDAR maps have much larger scale than the object-level voxel grids used by the existing 3D representation learning methods. The existing methods are too expensive and memory consuming for our task. We thus proposed to use a pyramid of max-pooling layers with different kernel sizes to overcome this challenge [26].

2.1.2 LiDAR to LiDAR map

The LiDAR maps used in this work are in the form of point clouds, which are built by accumulating and denoising individual LiDAR scans. In this section, we categorize and discuss existing point cloud registration methods by the corresponding compressive map formats. We refer to [15] for additional compression tools that focus on reconstruction accuracy – they can be applied on top of the following compressive maps, such as Octree [120] and bzip2 [1]. We also refer to [160] for global LiDAR localization works.

Raw Point Clouds

Iterative Closest Point (ICP) [10] registers two point clouds by iteratively finding the closest point pairs and computing the transformation matrix based on the found pairs. Since its debut in the early 90s, researchers have proposed a tremendous amount of ICP variants. ICP and its variants are still arguably the most widely adopted point cloud registration method in practical systems nowadays, despite its well-known drawback of being easily trapped in a local minimum.

The efficiency and accuracy of ICP variants mainly depend on the method of point correspondence search between source and target point clouds, and the quality of initialization. Greenspan and Yurick [56] proposed speeding up the correspondence search using a k-D tree. Generalized ICP (G-ICP) provides a probabilistic formulation that unifies point-to-point and point-to-plane ICP [124]. Modern off-the-shelf ICP tools such as PCL (Point Cloud Library) [111] and Open3D [167] are still vulnerable to local minima and require good initialization. Yang et al. [157] proposed Go-ICP that performs a global search to avoid the local minima at the cost of slow speed.

As for reducing the size of the raw point cloud maps to improve efficiency, Yin et al. [159] proposed to use the hit frequency as an indicator to prune LiDAR maps. Dubé et al. [40] proposed SegMap, which compresses semantic map segments with a 3D auto-encoder network and reconstructs raw point clouds for registration.

Feature Points

Compressing an input point cloud into representative key points with descriptors can potentially reduce the map size and improve the robustness of correspondence search. Feature point correspondences can be extracted by comparing the feature descriptors and the registration can be solved globally in a closed form using the Procrustes algorithm [32]. Rusu et al. [112] proposed Fast Point Feature Histograms (FPFH) as the descriptor for finding robust point correspondences. The noisy initial correspondences found by the descriptor matching can be filtered by robust methods such as RANSAC and TEASER++ [156].

Deep networks can be used to detect feature points and extract descriptors from raw point clouds. Wang et al. [147] used attention-based modules and the information from the other point cloud to learn the feature descriptors and the correspondences. Choy et al. [31] proposed Fully Convolutional Geometric Features (FCGF), which uses a 3D sparse fully-convolutional network to extract per-point descriptors, and a follow-up work [32] uses a 6-D sparse fully-convolutional network to predict point correspondences. Bai et al. proposed D3Feat [7] that uses KPConv [132] to extract dense point features, and trains point features by distance-learning losses for robust matching and importance scores. Points with low importance scores are pruned to compress the map. Fischer et al. proposed StickyPillars that uses a pillar encoder and a positional encoder to extract local LiDAR descriptors, and learn the apply the self and cross attention mechanism to obtain descriptors [46, 115].

Recently, transformers are used for learning LiDAR [73] or image-LiDAR fused [8, 165] features mostly for object detection and tracking. In contrast to the sparse convolution methods that only consider local neighborhood for descriptor computation, a transformer network increases the effective receptive field for point descriptors and performs better for faraway LiDAR points [73] (where the point cloud is sparse and not many neighboring point to compute descriptors from).

Shape Models

The local point cloud structure can be represented by compressive shape models, and registration can be performed without recovering the raw points. GMM-based methods use Gaussian models to approximate the local shape of point clouds and perform GMM-to-GMM or point-to-GMM registration using the EM algorithm [96]. Normal Distributions Transform (NDT) based methods perform efficient registration between NDT models [11]. Eckart et al. [43] proposed a hierarchical, anisotropic GMM tree for coarse-to-fine registration. Gao and Tedrake [49] proposed FilterReg that accelerates the EM algorithm by formulating the E-Step as a 3D filtering problem. Yuan et al. proposed DeepGMR [163] that replaces the E-Step using an end-to-end trainable network. The NDT representation can be combined with semantic labels to form a more expressive map representation [86]. If knowing the structure of environment beforehand, pre-defined shapes such as planes [169] can be use (e.g. align the ground planes from the scan and the map).

Without keeping the local structure, PointNetLK [5] compresses a whole point cloud into a single feature embedding using PointNet and performs direct feature registration with the feature embeddings. On the other hand, cylindrical range image of LiDAR scans can also be used for registration [38]. The intensity information is also proven to be useful in LiDAR-based localization [144]. It would be interesting to explore the role of intensity in LiDAR map compression as a future work.

2.1.3 Image to SfM map

A common image localization pipeline consists of two main steps. First, image-retrieval based global localization is performed to obtain an initial pose estimate, and then local feature matching is used to get an accurate final pose [63].

Classical Pipeline

Traditionally, the key point descriptors extracted during mapping are reused for local registration. After obtaining an initial pose from image-retrieval [53, 63, 140], we can match the key point descriptors extracted from the query image to the those in the nearby mapping images [63]. This approach avoid cross-modality (3D-2D) matching with the cost of larger map size – a 3D position corresponds to multiple key point descriptors that all need to be stored in the map. Also, if the scene changes after map collection (e.g. for long-term localization), the mapping and query descriptors might not match.

Robust Feature Learning

Many previous works have attempted to solve the long-term visual localization problem by finding robust feature descriptors against environmental changes [136] (such as day-night, lighting conditions, and seasonal changes). Concrete examples include R2D2 [108], SOSNet [133], PixLoc [116] and [2]. Some methods look into the dynamics of visual features (and the corresponding physical environment) such as persistency [42] and repeatability [37]. Besides learning robust features, some works also attempt to overcome the environmental challenges by finding common information in 2D and 3D, such as semantic information [134, 135] and predicting depth from query images [103]. In our work [27], instead of finding robust features, we focus on sparsifying the SfM map globally by taking the whole map graph structure into consideration.

Graph Neural Networks

The SfM map is essentially a heterogeneous graph that contains 3D map points and camera views as nodes, and the visibility links between camera views and 3D map points as edges. This makes apply graph neural networks (GNNs) [58] a natural fit for learning features from an SfM map.

Graph neural networks have been applied to a variety of learning tasks with irregular data structures, such as citation graphs [139] and image visibility graphs [125]. An important advantage of GNNs is the ability to handle heterogeneous data [146]. In our work, we represent the various information in SfM maps with heterogeneous graphs and extract features with a GNN. Recently, attention-based networks have shown strong performance in feature extraction from not only sequential data [138] but also graph structures such as 2D-3D matching [115]. Inspired by these works, we investigate the combination of heterogeneous GNN and attention, and demonstrated better final performance than the baselines [27].

2.2 Map Compression

For a map that contains redundant information of a world, the goal of map compression is to select the most valuable subset by removing unimportant parts, or by representing existing data with more compact representation.

2.2.1 Feature Clustering

Clustering techniques have been used in compression for decades. In [99], Oehler and Gray proposed to use Vector Quantization (VQ) to compress and classify medical images. Agustsson et al. [4] proposed a learned VQ to compress and decode the latent representation in an auto-encoder structure. Wei et al. applied task-orientation compression to form a 2D binary map [148]. In our work, we performed K-means clustering on the learned map features and only store the per-voxel centroid index as the map feature [26]. This is more compact than the binary code used in [148]. Also, the 2D to 3D registration problem we solve is more complicated and challenging than the 2D to 2D registration setting in [148] since it requires backpropagation through projection.

Besides feature compression, the clustering technique can al so be used to reduce neural network model size. In [54], Gordon et al. presented that by applying K-means clustering to NeRF weights, it largely reduces the network size with only small performance loss.

2.2.2 Point Selection with K-Cover

In previous image localization works, it is common to assume that the map contains all the possible camera positions, and formulate the map compression as a K-Cover problem, which encourages each possible camera position (the key frame location in the map) to observe enough 3D points for performing robust PnP during localization under a total point number budget. The K-Cover problem is then solved using various techniques: a probabilistic approach [16], Integer Linear Programming (ILP) [41, 84] and Integer Quadratic Programming (IQP) [41, 91, 101]. A hybrid map and hand-crafted heuristics were also used to determine the importance of map points [14, 83, 97]. Grid-based sparsification can be used together with K-Cover to improve efficiency [165].

The existing K-Cover based methods work well in a static world but suffer from performance degradation in vastly dynamic environments where many of the visibility edges in the map are outdated and invalidated. We proposed using this K-Cover concept together with robust feature point selection to perform map point selection for dynamic environments [27].

2.3 Neural Radiance Field

Neural Radiance Fields (NeRF) [93] is an implicit neural representation trained by overfitting an MLP network to a set of posed 2D images, and can be used to render novel views from complex 3D scenes. The MLP takes a camera view direction and a 3D position as input and predicts the corresponding color and density. When given a novel camera pose and intrinsics, a NeRF system draws rays from the query camera center through its virtual image pixel positions into 3D space, sampling 3D points along the rays, and accumulates the predicted color and density of the 3D sample points for each ray to obtain color values for each pixel. Given proper training data, a NeRF system can render high-quality synthetic images with realistic visual appearance and reasonable depth [9, 47, 76, 87, 93, 95, 102].

2.3.1 Large-Scale NeRFs

Here we focus on large-scale outdoor environments. Since it is inefficient to use a global MLP [93] to encode a large space, existing work leverages the divide-and-conquer approach – dividing the space into small parts such as street blocks [131] or voxels [59, 81, 106, 161], and assigning localized embeddings to represent the small parts.

Existing methods have also shown that using depth priors can significantly reduce the required number of source view images for NeRF [34, 76, 109]. Comparing to predicted depth [76] and SfM point clouds [34], LiDAR measurements are more robust and can better cover the geometry of texture-less regions where depth values are not well-constrained by photometric information. Rematas et al. proposed using LiDAR depth as supervision [107], and Carlson et al. proposed using trainable occupancy grid to assist ray sampling locality [18], but both [107] and [18] still use global MLPs. On the other hand, PointNeRF [154] proposed a point cloud based neural radiance field with localized embeddings, and greatly improved NeRF sampling locality and convergence speed. NPLF [100] aggregated point features into ray features with self-attention mechanism. In our work, we look into ways to improve [154] for challenging outdoor datasets and explore more practical applications.

A potential alternative system design for LiDAR-assisted NeRF is to follow NPLF [100]. NPLF aggregates point features into ray features with self-attention mechanism instead of explicit distance-based weights and volume rendering like ours and PointNeRF [154], and was trained without LiDAR depth loss. The attention mechanism provides more flexibility than the explicit method, and could potentially better overfit the training views. However, we expect the explicit method by PointNeRF to follow LiDAR geometry more faithfully.

2.3.2 Conditional GANs

Generative Adversarial Networks (GANs) have been applied to support NeRF in different ways [23, 72, 90, 98, 123]. For example, generative models were used to represent individual objects that could be combined into a full image by controlling object positions [98]. However, in general outdoor scenes, many objects are not labeled and thus cannot be easily segmented and represented with individual generative models. Previous works also attempted to perform novel view synthesis with 3D-aware GANs [35, 72, 90, 123], while existing 3D-aware GANs are limited to simple geometry such as small objects or faces and cannot be directly applied to general scenes.

On the other hand, 2D conditional GANs can learn the image translation between two distributions and produce visually realistic appearance [65, 168]. In contrast to vanilla GANs that demands many training data, cGANs benefit from the conditional input and can be trained on much fewer data, such as images captured by sonar and tactile sensors [80, 128]. GANcraft [59] applied a cGAN to translate semantic segmentation images into realistic images. In our work, the dense semantic labels are not available, but we also leverage a 2D cGAN to refine the volume rendering output [28].

The advantage of using cGANs to generate realistic-looking images from real-world datasets has been well-proven in many previous works [59, 70, 75, 105, 143, 168]. On the other hand, we would like to point out other potential alternatives. Recently, diffusion models also showed impressive performance in image synthesis tasks [36, 113, 114, 145]. Generally speaking, existing diffusion models require more training data and higher computation cost than cGANs. In contrast to the one-step image refinement cGAN used in our work [28], the application of iterative conditional diffusion models to real-world image quality refinement would be an interesting direction to explore. It is worth mentioning that InfiniCity [79] demonstrated a 2D-3D hybrid approach that generates synthetic voxel grids to perform city-scale voxel-based neural rendering. It would be interesting to see how this 2D-3D hybrid approach can interact with our real-world LiDAR measurements in the future.

Chapter 3

Map Compression for Image-to-LiDAR Registration

In this section , we address the problem of image registration to a compressed 3D map [26]. In contrast to LiDAR-to-LiDAR registration, camera sensors are much cheaper and suitable for low-cost and on-device applications. Unlike the previous work that learns relative pose by extracting 2D features and comparing projected depth and camera images [20], we developed HyperMap (Fig. 3.1), which offloads online depth map feature extraction to offline 3D map feature computation for the 2D-3D camera registration task through end-to-end training.

3.1 Introduction

In this work, we propose a novel strategy to learn on-map convolutional features to compress the map and preserve the registration performance. Given a noisy initial camera pose, our method predicts the relative 6 degree-of-freedom (DoF) pose of the camera by comparing the image captured by the camera to a virtual feature image created by projecting a 3D feature map to 2D. Although the methods that perform localization by comparing LiDAR scans collected in real-time to the HD map usually outperform the camera-based methods in terms of localization error [71], LiDAR sensors are substantially more expensive than cameras. Our solution leverages the robustness of LiDAR sensors in the offline map building process and then relies solely

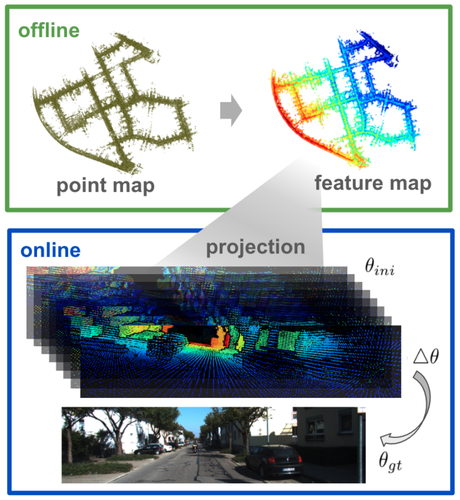


Figure 3.1: HyperMap workflow. Our network predicts the relative camera pose by comparing the camera image to a high-dimensional feature image created by projecting our feature map to 2D using the noisy initial pose.

on cameras to perform online pose registration. This design significantly reduces the online system cost while still leveraging the strengths of LiDAR sensors. In this work, we assume an approximate pose is known, which is reasonable since GPS is very common in modern devices and the existing global localization or vehicle pose estimation methods can be used as our input.

The registration of 2D camera images to a 3D point-cloud map is non-trivial due to the inherent difference in the modalities. Prior works have tried to solve the problem by projecting depth information from the 3D map into 2D to form a depth image from which features are then extracted to enable comparison with the observed camera image [20, 22, 30]. We refer to the decision to perform projection before feature extraction as "*early projection*" in this work. We propose instead the use of "*late projection*", a method which precomputes and compresses the 3D features on the voxelized point cloud map and then performs projection for subsequent alignment with the captured 2D RGB image. Our approach utilizes sparse 3D convolutional layers to extract features from the HD point cloud map [31, 32, 55, 167]. The sparse convolutional layers enable us to process large point clouds efficiently. Compared with current state-of-the-art methods [20, 22], which uses "*early projection*", our method compresses the learned features and reduces the required map voxel resolution significantly, and thus reduces $87 - 94\%$ of map size with comparable performance.

The primary contributions of this work is as follows:

- We propose "*late projection*" in contrast to "*early projection*" for the 2D-3D registration task. Our late projection strategy precomputes and compresses the 3D map features offline before online projection, which we refer to as a "HyperMap" due to the use of hypercolumn features [60].
- The proposed HyperMap outperforms the baseline in map size significantly while maintaining comparable or slightly better performance. Although we focus 2D-3D registration in this work, we believe that the concept of "*late projection*" can be extended to and potentially benefit other map-related tasks.
- In addition, we propose an efficient occlusion-handling layer that enables back-propagation from a projected feature image to 3D convolutional layers on a large-scale sparse point cloud map. This occlusion-handling layer is crucial to the scalability of our proposed HyperMap.

3.2 Method

In the proposed pipeline (Fig. 3.2), we first perform offline 3D sparse convolution to extract and compress the voxelwise hypercolumn features for the whole map. Then at run-time, we project and decode the compressed map features to the rough initial camera pose to form a virtual feature image. A Convolutional Neural Network (CNN) is then used to predict the relative pose between the camera image and the virtual feature image. In addition, we propose an efficient occlusion handling layer, specifically designed for large point clouds, to remove occluded points in projection. Our experiments on synthetic and real datasets show that, by moving the feature computation load offline and compressing, we reduced map size by $87 - 94\%$ while maintaining comparable or better accuracy.

3.2.1 Map Feature Extraction

To perform 3D feature extraction on the LiDAR map efficiently, we voxelize the 3D point cloud map in high resolution to extract the 3D convolutional features and then

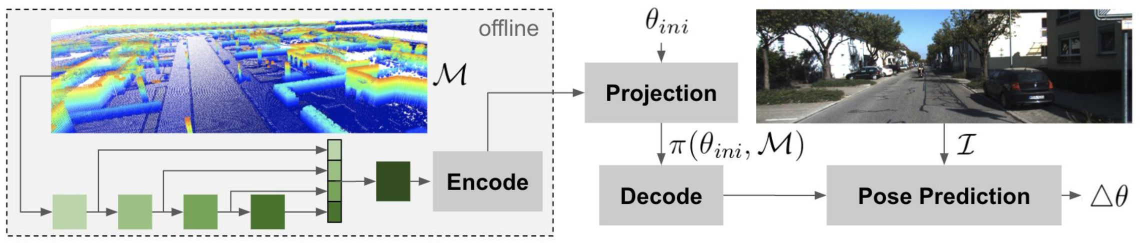


Figure 3.2: System architecture. The sparse convolutional layers are shown by green boxes. We only store the compressed map.

downsample, for which we adopt the 3D sparse convolutional filter [31, 32, 55, 167] due to its great scability and efficiency. We extract the convolutional features using a set of 3D sparse convolutional layers, which only operate on the occupied voxels and are suitable for sparse point cloud data from sensors like LiDAR.

In order to capture features with different frequencies, we apply the hypercolumn [60] concept to 3D feature extraction. We use stride 2 for the first 3D convolutional block and 1 for all the other blocks. The receptive field of each layer expands as more convolutional layers are applied, and the feature dimension also increases correspondingly. Afterwards, we combine the multiple activations to form a hypercolumn feature vector for each occupied voxel to preserve both the precision of earlier layers and the capacity of later layers. The final voxel resolution was thus reduced by ratio two due to the stride 2 in the first block.

At training time, we first voxelize the whole raw point cloud map, crop the local map region using the initial pose, extract 3D features in the map coordinate frame, and then transform the cropped feature map to the camera coordinate frame. Afterwards, n layers of 3D sparse convolutional filters are applied to the voxelized local map, and the feature output from the n convolutional layers are concatenated to form a high-dimensional hypercolumn feature vector.

For a voxel v_i in the map, the corresponding hypercolumn feature vector is first compressed to a lower dimension feature $f_i \in \mathbb{R}^m$ (dimension $72 \rightarrow 16$) using another 3D sparse convolutional layer. After trained end-to-end, we apply K-means algorithm to all the f_i in the map to obtain k centroids, and compute the cluster index d_i of each voxel (As shown in Fig. 3.3, we use $k = 16$ in our experiments, so we only need 4 bits to represent the centroid index, $d_i \in 0, 1, 2, ..., 15$). We then project the cluster

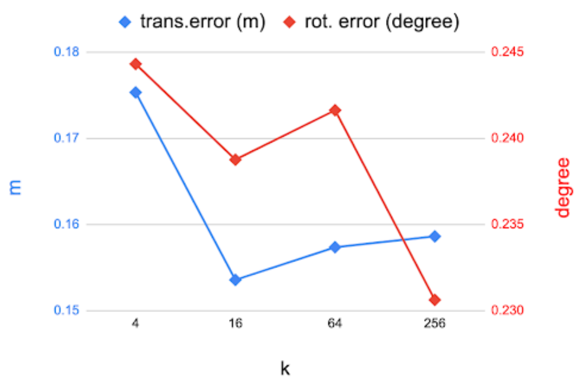


Figure 3.3: Accuracy plot of using different number of centroids in K-means with CARLA dataset. We observed no obvious benefit of using more than 16 centroids.

index d_i to form a 2D virtual feature image, and recover the original feature f_i from d_i using the corresponding K-means centroids. Notice that map feature projection required retrieving the map feature data from the storage and thus projecting d_i is cheaper and faster than projecting f_i due to its small size.

In the map projection step, we project the voxel grids to form a depth map and concatenate the depth map to f_i as an additional channel, so the final projective virtual feature image has $m+1$-dimensions. This feature precomputation step reduces the required voxel resolution while preventing the performance drop. We use kernel size 3 for all the 3D sparse convolutional layers.

3.2.2 Occlusion Handling

The compressed map features are projected and decoded to form a virtual feature image using the camera intrinsics and the given initial pose. However, because of the nature of sparse point clouds, the occluded points may appear in the virtual feature image if not handled. To remove the occluded points, we design a maxpooling pyramid inspired by the point cloud occlusion filtering described in [20]. Our occlusion handling layer is very efficient and is suitable for large-scale point clouds since it only utilizes the max pooling layers.

We use the voxel size to approximate the occupied neighborhood of the map points in 3D space, and projection of the occupied neighborhood should only contain the projections of the map points that are closer to the camera than the voxel center (with smaller depth value). This means that if a projective map point has some nearby pixels with smaller depth, it is likely that this voxel is occluded. We use efficient maxpooling filters to simulate the 2D occupied neighborhood. In order to apply the maxpooling layers, we first make the projective depth negative and set the empty pixels to the maximum negative depth value. The pixels with smaller depth values originally would be larger after this transformation, and thus will be kept after the maxpooling operation. Afterwards, we recover the original image by setting the empty pixels back to zero and inverting the sign of the depth map. The output, maxpooled depth map, is noted as $M_r(\mathbf{p})$ with kernel size r at pixel position $\mathbf{p}$.

Let $D(\mathbf{p})$ and $R(\mathbf{p})$ be the depth map and its corresponding occlusion filter kernel size map, and f be the focal length. The map of the occlusion filter kernel size (in pixel) can be computed from the fixed voxel size in the map:

$$R(\mathbf{p}) = \frac{\text{voxel size} \times f}{D(\mathbf{p})} \tag{3.1}$$

Afterwards, we find the pyramid level with smallest $M_r(\mathbf{p})$ among all levels for each pixel, denoted as:

$$r_{min} = \arg\min_{r} \; M_r(\mathbf{p}). \tag{3.2}$$

If r_{min} is larger than the $R(\mathbf{p})$ at this pixel, it means that this pixel is occluded by a nearby pixel with smaller depth value and the corresponding feature value should be set to zero. Let $F(\mathbf{p})$ be the virtual feature image. The final virtual feature image is computed by:

$$F_{final}(\mathbf{p}) = \begin{cases} 0, & \text{if } \arg\min_r \; M_r(\mathbf{p}) - R(\mathbf{p}) > \delta \\ F(\mathbf{p}), & \text{otherwise} \end{cases} \tag{3.3}$$

We choose $\delta = 0.5$ so the occlusion filter is only effective when the occluded points are far away from the visible point. If several layers in $M_r(\mathbf{p})$ have the same pixel value, which happens when all the maxpooling layer outputs are dominated by a close-by nearer point, we pick the smallest r among them. Results are shown in Figure 3.4.

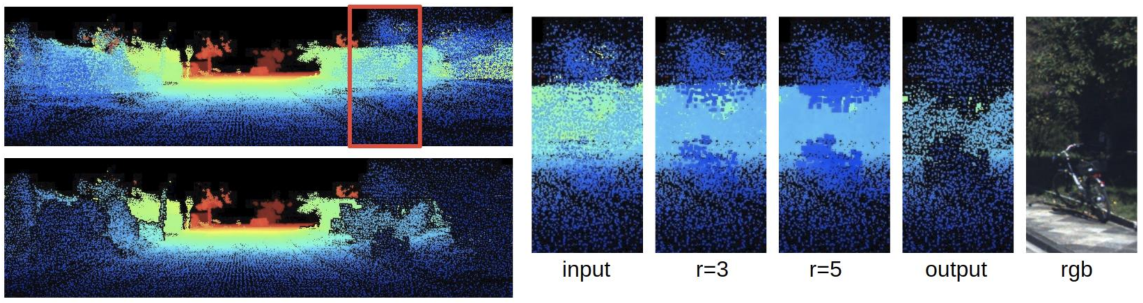


Figure 3.4: Occlusion handling. We use a max pooling pyramid to implement an occlusion handling layer. The above figure shows the effect of applying a maxpooling kernel with several different sizes, where the occluded pixels are set to zeros (shown in black), and the input and output of the occlusion handling layer. Larger depth values are shown in red and small depth values are shown in dark blue.

3.2.3 Camera Pose Prediction

Given the virtual feature image and the RGB camera image, we regress the relative camera pose $\triangle\theta$ using a CNN following [20]. We use the image feature extraction branch of PWCNet [129] for RGB feature extraction and simply replace the dimension of the first convolutional block in the depth feature extraction branch with our projective map feature dimension. A correlation filter is then used to match the features from the RGB image and the virtual map feature image. Several fully connected layers are used to predict the translation in xyz directions and the quaternion for rotation to represent 6-DoF camera pose. We add one additional tanh layer as an output layer to constrain the range of predicted translation and rotation. For training, we use Smooth L1 loss and quaternion angular distance loss as proposed in [20].

3.3 Experiments

In this section, we describe the evaluations on CARLA synthetic dataset [39], KITTI Odometry dataset [51], and Argoverse Tracking dataset [24]. We choose CMRNet [20] as our baseline and compare with it in 0.1m, 0.2m and 0.4m voxel resolutions. We use $n = 4$, $m = 16$, $k = 16$, and a five-level occlusion pyramid (maxpooling kernel $r = 3, 5, 11, 15, 23$) in all experiments (the details of the parameters are in Section 3.2.1).

3.3.1 Data Preparation

For the CARLA dataset, we used the official data collector to collect single camera sequences with ground truth poses. We collected seven sequences for the training set (14755 frames) and two sequences for the validation (4228 frames). The validation set contains weather conditions that do not exist in training set, shown in Fig. 3.7. The point cloud map (Town01) was downloaded from the official repository.

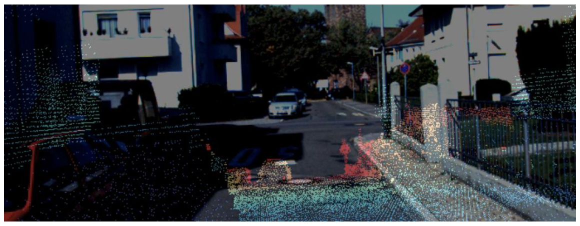
As for the KITTI Odometry dataset, we used the LiDAR maps, the ground truth poses and the initial poses for validation set provided by the authors of CMRNet [20] We used the sequences 03, 04, 05, 07, 08 and 09 in the KITTI Odometry dataset as the training set (10581 frames) and the randomly downsampled sequence 00 as the validation set (1500 frames). We excluded sequence 06 due to the artifacts in the generated map. The validation map does not overlap with training maps except for only 200 frames. We used SLAM poses from [20] as the ground truth since the KITTI ground truth poses are noisy. As mentioned in [20], the ground truth poses in KITTI dataset caused map inconsistency in loop closures, so we used the ground truth poses provided by CMRNet authors as well, which was optimized by loop-closure SLAM method as described in [19, 20].

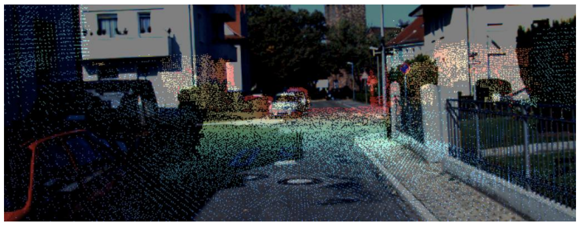
We built the Argoverse maps by accumulating the LiDAR scans using the provided ground truth poses. We uniformly downsampled the original train and validation splits as the training set (9328 frames from 85 logs), and also downsampled the original test split as the test set (1599 frames from 24 logs). We removed log 3373 and 7d37 from the training set because the ego vehicle was surrounded by large buses.

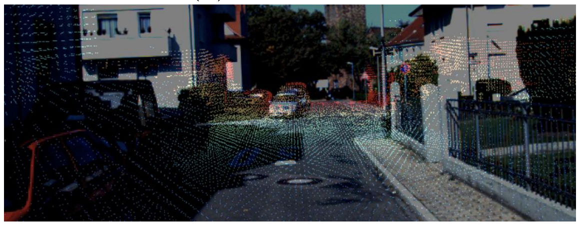
We downsampled the maps using voxel resolutions $0.1m$, $0.2m$, and $0.4m$ for the baseline experiments, and used the $0.2m$ resolution as the input of our HyperMap. The final HyperMap resolution is $0.4m$. To simulate erroneous initial pose, we added translation noise within [-2m, +2m] in xyz directions, and rotation noise of [-10°, +10°] about xyz axes applied in xyz order following [20]. The initial poses were generated online in training time and fixed in test time.

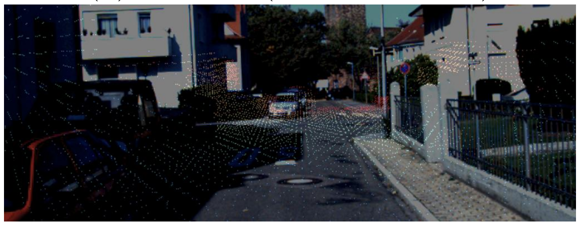
Implementation Details

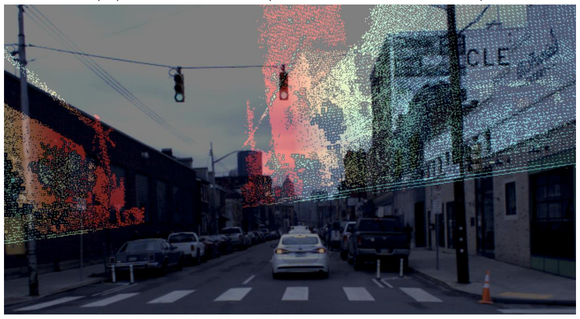
Aiming for a fair comparison, we integrated the CMRNet into our pipeline, so that the only difference in the experiments was the network itself. We added a scaled tanh

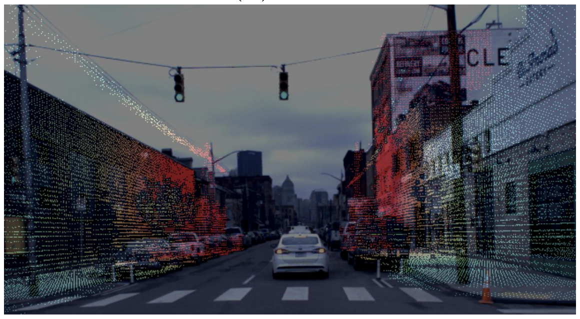

(a) Initial pose

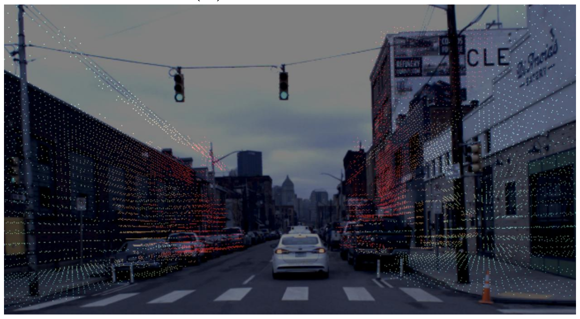

(b) Baseline (0.1m voxel size)


(c) Baseline (0.2m voxel size)


(d) Ours


(e) Initial pose


(f) Baseline (0.1m voxel size)


(g) Baseline (0.2m voxel size)

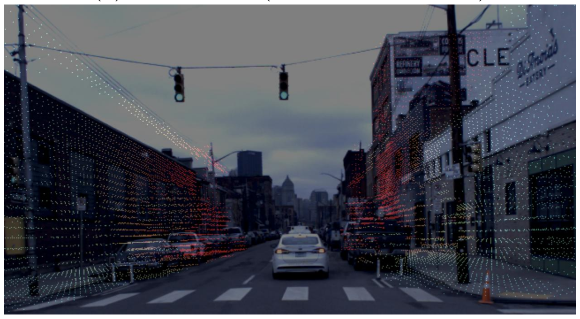
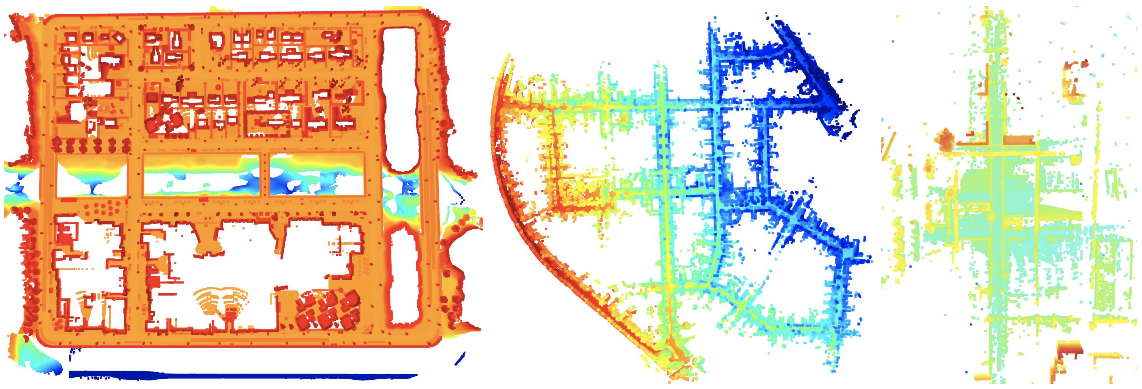
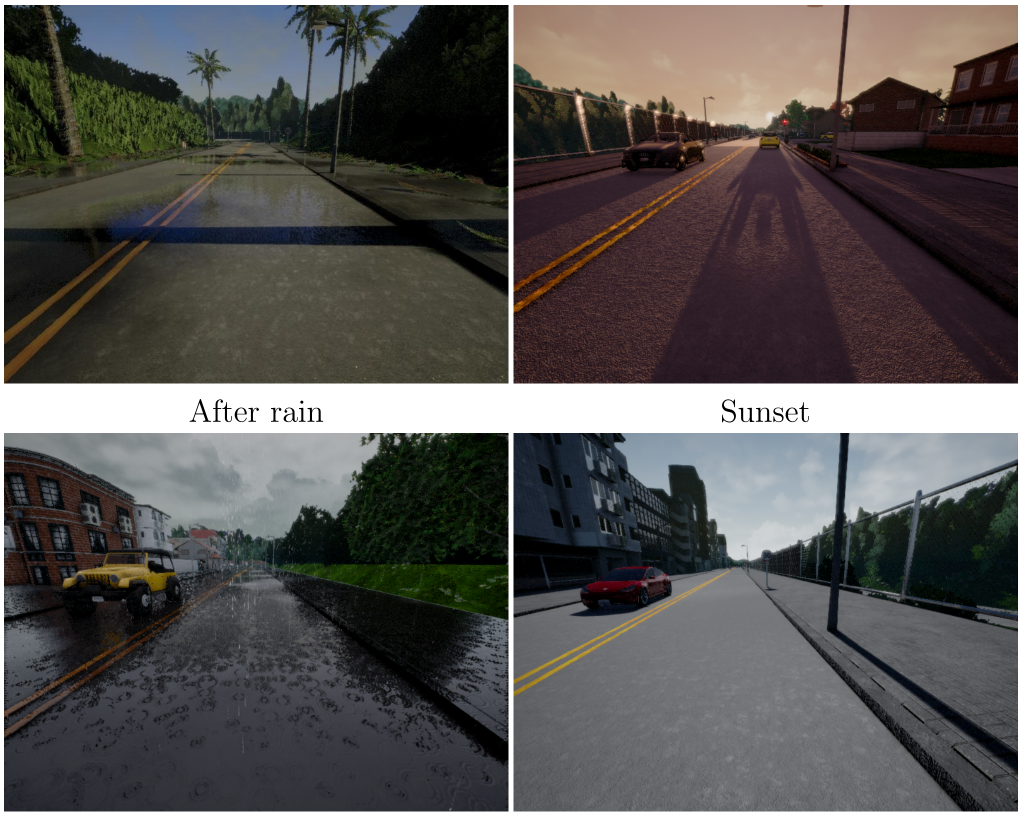

(h) Ours

Figure 3.5: Visualizations of the projected depth maps from KITTI Odometry dataset (a)-(d) and Argoverse Tracking dataset (e)-(h). Further distance is represented by red, closer distance is represented by blue. Best viewed in PDF file.



CARLA KITTI 00 Argoverse c691

Figure 3.6: Visualization of maps



Rain Sunny

Figure 3.7: Visualization of the maps and the CARLA weathers.

layer to the CMRNet implementation at output to leveraging the prior knowledge of the known noise range, as we did in our HyperMap. We split the training process into two stages. First, we cropped the local map around initial camera pose with radius 50m and voxelized it and then applied the 3D sparse convolutional layer to the local voxelized map to extract map features. Afterwards, the extracted map feature was projected to form a virtual feature image for pose prediction and initial training. Second, after the map feature is well-trained, we applied the pretrained sparse convolutional layers to the whole voxelized map to get the map features f_i, using K-means to get the centroid index d_i for each voxel, and only store the d_i in the map for the map size comparison. Afterwards, we fixed the map features, only refining the pose prediction network until convergence. The refinement step helped to compensate the compression error induced by K-means. Given the scale of our maps and the efficiency of the sparse convolutional networks, we were able to process the whole map offline on our lab server without splitting it into submaps for the experiments. This approach is scalable to larger maps with a divide-and-conquer approach since the convolutional filters are translationally invariant and the receptive fields are limited.

We implemented all the models in PyTorch. All the models are trained and timed on an Intel(R) Xeon(R) CPU E5-2660 v4 @ 2.00GHz machine with GeForce GTX TITAN Xp GPU. We train all the models using learning rate 10^{-4} and batch size 40 with Adam optimizer. The occlusion handling layer takes about $1ms$ and the pose prediction takes about $14ms$ on our machine for KITTI odometry dataset.

3.3.2 Performance

We observed that our HyperMap has comparable or better accuracy in both the synthetic and real-world datasets, especially in translation, and much smaller map size than the baseline. Our method, as shown in Tab. 3.1, outperforms the $0.4m$ baseline significantly and even outperforms the $0.1m$ baseline in the Argoverse Tracking dataset where the baseline map size is more than seven times larger. We adopt a sparse representation to store the map. The map size is computed as the total storage required to store all the 3-dimensional indice (2-byte integer coordinates) of the occupied voxels (map points) and the corresponding voxel features (4-bit for the

Table 3.1: Quantitative comparisons. Overall, we reduced the total map size by $87 - 94\%$ with comparable accuracy.

Dataset	Model	Voxel size(m)	Trans.(m)	Rot.(°)	Map size ($bytes/m^2$)
CARLA	CMRNet + tanh	0.1	0.16	0.30	524.2
	CMRNet + tanh	0.2	0.18	0.29	193.5
	CMRNet + tanh	**0.4**	0.24	**0.24**	**52.5**
	HyperMap	**0.4**	**0.15**	**0.24**	56.7
KITTI	CMRNet + tanh	0.1	**0.45**	**1.35**	1471.2
	CMRNet + tanh	0.2	0.47	1.40	330.3
	CMRNet + tanh	**0.4**	0.63	1.97	**75.7**
	HyperMap	**0.4**	0.48	1.42	81.8
Argoverse	CMRNet + tanh	0.1	0.60	1.35	717.1
	CMRNet + tanh	0.2	0.61	0.95	282.7
	CMRNet + tanh	**0.4**	0.65	0.95	**89.0**
	HyperMap	**0.4**	**0.58**	**0.93**	96.0

16 K-means centroid indices). The baseline map only contains the voxel indices and no feature. Although increasing the voxel size reduces the map size effectively, the corresponding baseline performance drops, as shown in Tab. 3.1. We used $bytes/m^2$ as the map size unit since the maps in our scenario are usually very flat. Our method compressed the local statistics into voxel features, generated feature maps with voxel size $0.4m$, and reached comparable performance with the baselines using smaller voxel sizes, at the cost of small storage overhead for storing the features.

3.4 Discussion and Conclusion

In this work, we demonstrated the valuable potential of the proposed offline map feature preprocessing. It is possible to apply additional compression methods on top of our results to further compress the maps. The map compression advantages from our method would still be valid in this case. We have so far demonstrated that the proposed approach is effective with voxel-based downsampling. It would be interesting to explore different point cloud downsampling methods, such as uniform downsampling or selective downsampling in the future. In addition, although not the focus of this work, it is possible to further improve the performance of HyperMap

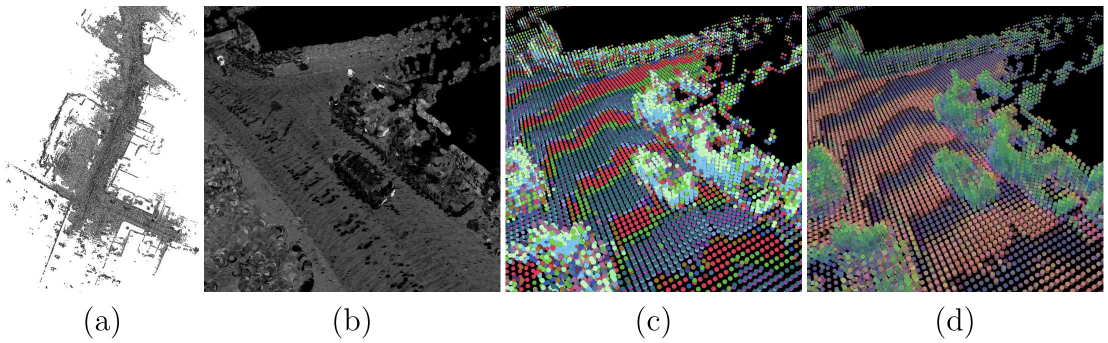


Figure 3.8: Visualizations of (a)(b) the raw point cloud map, (c) the K-means centroids stored on the map, and (d) the reconstructed map features from the K-means centroids and the stored indices in Hypermap. Notice that the LiDAR sweep pattern from the point cloud map generation process is also captured by the feature extractor.

using an iterative approach [20, 22]. In addition to the methods used in this work, other types of features, such as semantic labels, loop-closure features, and other 3D representations (like PointNet [104], FCGF [31]), can be added to the offline process and potentially improve performance. The advantage of the lowered voxel resolution is obvious in reducing map storage and the online query and processing time. Furthermore, the benefits enabled by the "*late projection*" paradigm opens the door to many possibilities in algorithm and system design, such as iterative optimization, high-speed localization, and cheaper on-board computers. We look forward to investigating other applications and methods that take advantage of the "*late projection*" way of thinking.

Chapter 4

Map Compression for LiDAR-to-LiDAR Map Registration

In this chapter, we perform a map compression benchmark for deep-learning based and classical LiDAR-to-LiDAR map registration methods [25]. Our results reveal the trade-off between map size and registration accuracy of the evaluated methods.

4.1 Introduction

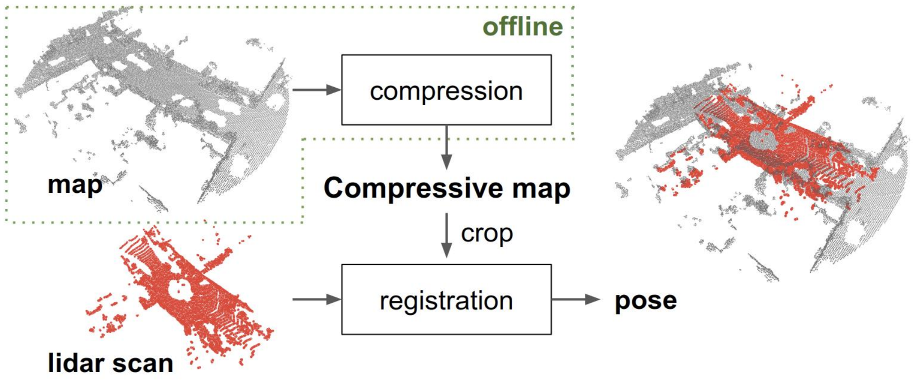
Maps are essential for modern autonomous driving systems. A map with rich prior knowledge provides valuable, offline-refined information that is not observable by online sensors, and thus improves the system performance. Modern maps, such as the HD maps used by autonomous vehicles, mostly contain high-quality dense 3D models and semantic labels. However, these dense 3D models require vast storage space and cause extra online data processing overhead.

The dense 3D model is mainly used to achieve accurate sensor-to-map registration, which is a crucial task for the autonomous vehicles to re-localize against the map when pose estimation fails, and also to reduce pose drifting errors in large-scale environments. Martinez et al. proposed a benchmark for retrieval-based localization methods because the dense HD maps are too expensive to collect and build at scale [88]. However, without the prior knowledge from the map, such retrieval-based methods require much training data and generalize poorly in unseen environments.

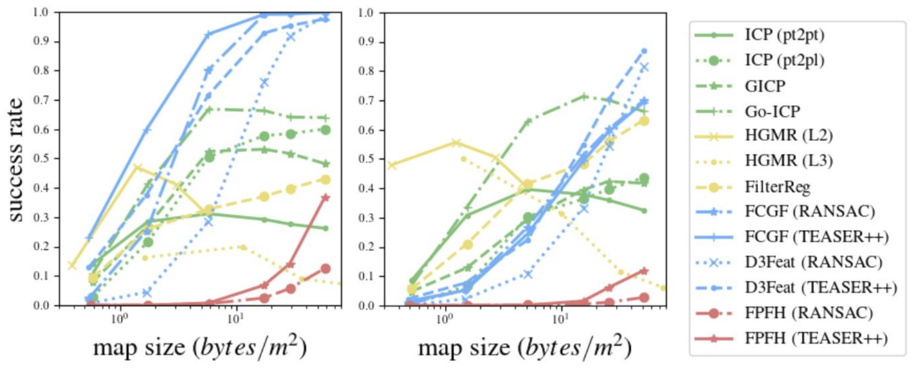
In practice, the dense 3D models are unnecessary for the essential tasks in autonomous driving other than relocalization. For example, motion planning, motion forecasting, object tracking, and obstacle avoidance only require the sensor input and the semantic map labels with rough 3D information, such as lane directions and the bounding boxes of the traffic lights. Since other information in the HD map is much lighter in size, eliminating the need of dense 3D models in the sensor-to-map registration process would reduce the total HD map size significantly.

Although eliminating the need of the dense 3D model is desirable, it deserves more research attention. Most existing point cloud registration studies focused only on the accuracy and speed of registering two scans with similar data distributions, while the data distributions of a sensor scan and a map are very different. Relevant benchmarks evaluate the point cloud compression performance by reconstruction accuracy, not by sensor-to-map registration accuracy [15]. In fact, loading a perfectly reconstructed dense 3D model is unnecessary if accurate sensor-to-map registration can be achieved with a lighter map. Although some works have evaluated sensor-to-map registration against map compression ratio [7, 159] for the proposed specific data formats, there is no universal standard available for a fair quantitative comparison among different compressive map formats.

In this work, we focus on a popular setting – registering a 3D LiDAR *scan* to a 3D *map*, which is the most common configuration for the modern autonomous vehicles to perform sensor-to-map registration. The raw map in this case is a high-quality, dense, and large-scale point cloud built offline. We propose that a sensor-to-map registration algorithm should operate directly on a certain compressive map format, instead of the raw point cloud, to eliminate the need of storing and processing the original large-scale point cloud. We refer to this pipeline as *compressive registration* in the following. The proposed compressive registration pipeline, as shown in Fig. 4.1, has several advantages over the methods using raw point cloud maps: 1) The map feature can be pre-computed offline since it does not require any online input. 2) The online map data decompression, if needed, takes less time since it does not need to recover a dense 3D map. 3) It takes much less storage space and data transmission time. As a result, we are interested in the sensor-to-map registration methods that



(a)



(b)

Figure 4.1: (a) The system pipeline of the proposed compressive registration. We compress the map offline, and register the online LiDAR input to the compressive map. (b) The success rate trends of the evaluated methods under different map size budgets on KITTI Odometry Dataset (left) and Argoverse Tracking Dataset (right).

directly operate on compressive formats.

We propose the first benchmark for compressive LiDAR-to-map registration. Given initial inaccurate LiDAR pose estimations, we evaluated the LiDAR-to-map registration performance on various compressive maps, including raw points, hierarchical GMMs, and feature points, under different map size budgets. Our benchmark is challenging due to the different data distribution of the LiDAR scans and the maps. We design universal map size based metrics for quantitative comparison. Our results illustrate the different trade-off trends between map size and robustness of the recent deep-learning based methods and classical methods. We show that the deep-learning based methods performed the best under high map size budgets but might perform worse than the classical methods using simpler models under low map size budgets, depending on the local map structure. As an additional contribution, we analyzed the robust registration methods, RANSAC and TEASER++ [156] together with various 3D features and show that TEASER++ in general outperforms RANSAC. To summarize, our contributions are:

- We propose the first compressive LiDAR-to-map registration benchmark. Our benchmark evaluates the map compressibility, robustness, and precision, and can be applied to various map formats.
- We evaluated both recent deep learning based and classical point cloud registration methods, including raw point based, GMM based, and feature point based methods. Our quantitative results reveal the trade-offs made by different methods and provide a valuable reference for future research.
- We released the benchmark for the community to evaluate more methods conveniently in the future.

4.2 Method

4.2.1 Overview

We propose a universal benchmark for compressive sensor-to-map registration for various compressive map formats. The proposed compressive registration pipeline is illustrated in Fig. 4.1. In the pipeline, we first perform offline map feature computation and compression, crop the local map using a noisy initial pose, and then register an input LiDAR scan to the cropped compressive map. The LiDAR scan is converted into the corresponding format used in the evaluated registration methods, such as feature points or GMMs. Let $\mathbf{P}$ be the source point cloud and the input LiDAR scan, $\mathbf{Q}$ be the target point cloud and the cropped map, and $\mathbf{T} \in \mathrm{SE}(3)$ be the transformation matrix that comprises the rotation matrix and the translation. The problem of point cloud registration can be defined as:

$$\mathbf{T}^* = \arg\min_{\mathbf{T}} \ \mathcal{L}\Big(f(\mathbf{T}, \mathbf{P}), \mathbf{Q}\Big), \tag{4.1}$$

where $f(.)$ denotes the point cloud transformation function, and $\mathcal{L}$ denotes the cost function used in the point cloud registration method. The cost function $\mathcal{L}$ varies among different methods. For example, point-to-point ICP uses Euclidean distances between selected point pairs and point-to-plane ICP uses squared distance from a point to a paired local plane patch. For methods that operate on other formats instead of raw point clouds, denoting $\phi(.)$ as the general feature extraction function, Eq. (4.1) becomes:

$$\mathbf{T}^* = \arg\min_{\mathbf{T}} \ \mathcal{L}\Big(f(\mathbf{T}, \phi_p(\mathbf{P})), \phi_q(\mathbf{Q})\Big). \tag{4.2}$$

Notice that $\phi_p(.)$ and $\phi_q(.)$ are not necessarily the same. For example, one can register a raw point cloud to a GMM model.

We assume a noisy initial pose is available – in practice, an autonomous vehicle receives the GPS signal and performs pose estimation on-the-go. The map is stored in the world coordinate frame. Let the transformation from the local LiDAR frame to the world frame be $\mathbf{T}_l^w$ and ideally $f((\mathbf{T}_l^w)^{-1}, \mathbf{Q})$ would be aligned with $\mathbf{P}$. And the map feature extraction $\phi_q(\mathbf{Q})$ should happen in the world coordinate frame since the initial pose is not available in the offline map preprocessing step. Letting $\mathbf{T}_{ini}$ be an initial noisy estimation of $\mathbf{T}_l^w$, Eq. (4.2) can be rewritten as:

$$\mathbf{T}^* = \arg\min_{\mathbf{T}} \ \mathcal{L}\Big(f(\mathbf{T}, \phi_p(\mathbf{P})), f(\mathbf{T}_{ini}^{-1}, \phi_q(\mathbf{Q}))\Big). \tag{4.3}$$

4.2.2 Method Categories

We categorize registration methods by map data types and techniques used for compression. A list of related methods is shown in Tab. 4.1, whose attributes are explained as follows:

- **Map type**: the actual data format used for registration, such as raw points, GMMs, and feature points.
- **Data dimension**: the dimension of the used data format. For example, point-to-point ICP uses only xyz coordinates so the dimension is 3. Point-to-plane ICP and GICP use the additional 3D normals thus the dimension is $3 + 3 = 6$.

Table 4.1: A list of related registration methods and the corresponding categories.

Map type	**Method Name**	**Data Dim.**	**Deep**	**Global**	**Scalable**
raw points	ICP (pt2pt)[1] [10]	3			✓
	ICP (pt2pl)[2] [155]	6			✓
	GICP [124]	6			✓
	Go-ICP [157]	3		✓	
GMMs	CPD [96]	3			
	NDT [11]	9			✓
	HGMR[43]	10			✓
	FilterReg [49]	3[3]			✓
	DeepGMR [163]	5[4]	✓	✓	
feature points	FPFH [112]	36		✓	✓
	DCP [147]	515	✓	✓	
	FCGF [31]	35	✓	✓	✓
	D3Feat [7]	35	✓	✓	✓
	DGR [32]	35	✓	✓	
global embedding	PointNetLK [5]	1024	✓		
hybrid	LORAX [44]	1035[5]	✓	✓	

- **Global**: the method does not require a good initial pose.
- **Scalable**: the method is feasible for building a large-scale compressive map.
- **Deep**: the method is deep learning based.

Some methods are not considered to be scalable for practical reasons: Go-ICP [157] and CPD [96] are much slower than other methods when running with our LiDAR point clouds. The feature dimensions of DCP [147] and LORAX [44] are very high and lead to huge map size if we compute and store the features in the map. The sparse 6-D convolutional network in DGR [32] is not applicable to very sparse inputs when map size budget is slow. The PointNet backbones used by DeepGMR [163] and PointNetLK [5] are only suitable for small object-scale point clouds.

[1]ICP (pt2pt) represents point-to-point ICP

[2]ICP (pt2pl) represents point-to-plane ICP, which requires normal input.

[3]We evaluated the FilterReg version with fixed covariance and equal weights [49], so the data dimension is the same as raw points.

[4]DeepGMR uses weighted isotropic GMM formulation [163]

[5]This is the dimension of the super points used in LORAX. An additional ICP refinement with the dense raw point cloud is required by LORAX besides the super points [44].

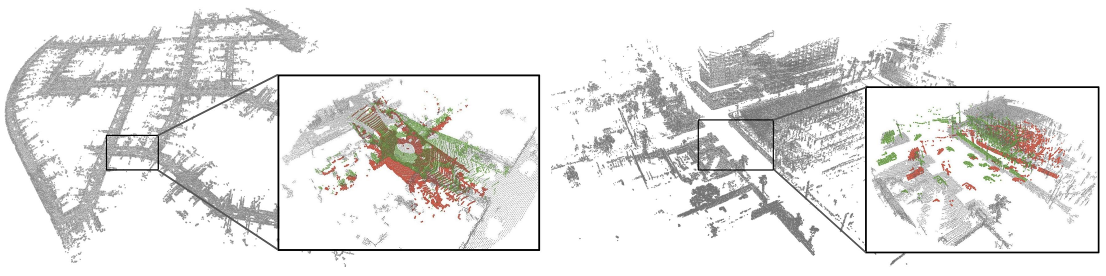


Figure 4.2: The maps and input LiDAR scans from (a) KITTI Odometry Dataset and (b) Argoverse Tracking Dataset. The input noisy poses are shown in red, and the ground truth poses are shown in green. We removed other vehicles using PV-RCNN for (a) and the provided driveable region map for (b).

4.2.3 Benchmark on Compressive Registration

A major difference between our benchmark and other existing evaluations is the asymmetry between the source (the LiDAR scan) and the target (the map). A LiDAR scan is sparser, noisier and contains moving objects (e.g. other vehicles), while a map is denser, pre-built, and refined by denoising and moving object removal. Please see Fig. 4.2 for the visualizations of our LiDAR scans and the maps.

To cover the initial error range in the real environment, we applied uniformly distributed noise within $[-10, 10]m$ to the xyz dimensions of translation and $[-10, 10]^\circ$ to the roll, pitch, and yaw rotation angles. This error range covers most of the possible GPS errors of a modern autonomous vehicle, according to [88]. To evaluate the pipeline in Fig. 4.1 with large-scale maps, we first preprocess the dataset into pairs of local maps and lidar scans. For each pair, We cropped a local map region within a $40m$ range around the initial pose, and then compress the local the map region to perform registration. The order of map cropping and compression does not affect the compression result for the compression methods used in this work. For the feature extraction methods such as FPFH [112], FCGF [31], and D3Feat [7], we precomputed feature extraction in the world coordinate frame, since the initial pose was not available when performing offline map compression.

4.2.4 Data Preparation

For this work, we focus on the autonomous driving scenario and prepared data from two real-world autonomous driving datasets: the KITTI Odometry Dataset [51] and the Argoverse Tracking Dataset [24] (KITTI and Argoverse for short). We aggregated the LiDAR scans to build a dense point cloud map. For KITTI, the provided ground truth poses are noisy, so we used the the poses estimated by SLAM [69]. We used the provided ground truth poses for the Argoverse. Considering that vehicles are the most common moving objects in the autonomous driving scenario, we removed vehicles form the maps. For KITTI, the vehicles were first detected by PV-RCNN [126] and then removed from the input LiDAR scans before building the map. For Argoverse, the vehicles were removed by pruning the driveable regions in the LiDAR scans before building the maps.

As for source clouds, we used the LiDAR scans from sequence 00 of KITTI and the test set from Argoverse. The KITTI sequences 03, 05, 07, 09 and the Argoverse training set were used to train the deep learning based methods. We applied a simple threshold-based ground removal to the input scans of Argoverse to match the map point distribution, as visualized in Fig. 4.2. The KITTI data contains 2271 scans and a map area of 4,678,598 m^2. The Argoverse data contains 1545 scans and a map area of 3,590,315 m^2. The map area is computed by the area of occupied regions at the m^2 resolution. A LiDAR scan of both datasets contain 64 bins and the FoV of a Argoverse LiDAR scan is wider (50°) than a KITTI LiDAR scan (26.9°).

4.2.5 Evaluation Metrics

We define evaluation metrics with the robustness and precision at different map sizes. The robustness is measured by *success rate* (also referred to as *recall*), and the precision is measured by the translation and rotation errors among successful samples. Because the total map area varies, we quantify map size with density $bytes/m^2$. Since the actual speed evaluation largely depends on the implementations and varies across different platforms, we refer the readers to the original papers for detailed speed comparisons.

Let $\mathbf{R} \in \mathrm{SO}(3)$ be a rotation matrix and $\mathbf{t} \in \mathbb{R}^3$ be a translation vector from $\mathbf{T}$. We measure the precision by:

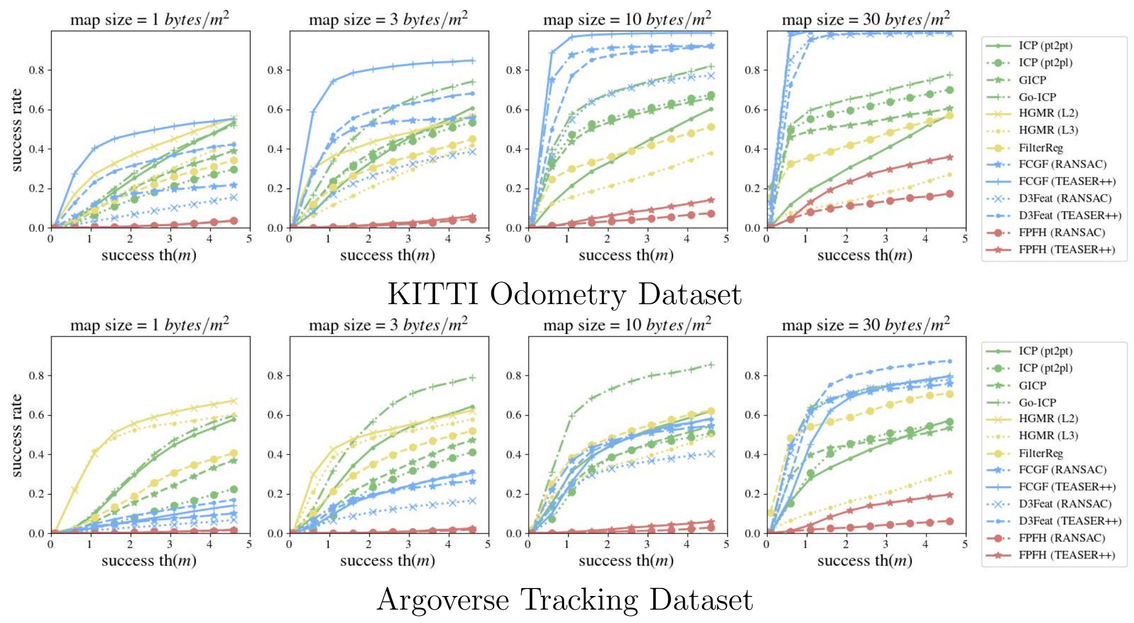


Figure 4.3: The success rate curves. We observed that FCGF (TEASER++) and HGMR (L2) outperformed all other methods in the case of map size = 1 $byte/m^2$ in KITTI and Argoverse.

- **Translation Error (TE)**: the median of the L2 distance between the translation vectors of the successful pairs:

$$\text{TE} = ||\mathbf{t} - \mathbf{t}_{gt}||_2^2. \tag{4.4}$$

- **Rotation Error (RE)**: the median of the rotation angle between the rotation matrices of the successful pairs:

$$\text{RE} = \arccos \frac{tr(\mathbf{R}\mathbf{R}_{gt}^T) - 1}{2}. \tag{4.5}$$

Here the subscript gt denotes the ground truth.

We measure the robustness by:

- **Success Rate (SR)**: the ratio of the pairs with both translation and rotation error lower than the assigned successful threshold. We choose the successful thresholds to be 2m for TE and 5° for RE as [7, 158].

We attach numbers to the metrics to represent the value under a map size budget.

Table 4.2: Results on the KITTI Odometry Dataset. Overall FCGF (TEASER++) and D3Feat (TEASER++) outperformed all other methods in robustness under all map size budgets.

Metric name	**SR01**	**TE01**	**RE01**	**SR03**	**TE03**	**RE03**	**SR10**	**TE10**	**RE10**	**SR30**	**TE30**	**RE30**
map size($bytes/m^2$)		1			3			10			30	
ICP (pt2pt)	0.19	1.11	2.28	0.31	0.82	1.38	0.30	0.74	1.22	0.27	0.76	1.12
ICP (pt2pl)	0.10	1.23	2.62	0.45	0.67	1.39	0.53	0.48	1.03	0.60	0.21	0.56
GICP	0.15	1.17	2.61	0.47	0.71	1.17	0.53	0.50	0.77	0.49	**0.09**	0.20
Go-ICP	0.22	1.23	2.66	0.62	0.68	1.34	0.67	0.51	1.02	0.64	0.27	0.60
HGMR (L2)	0.34	0.82	**1.37**	0.33	**0.22**	**0.34**	-	-	-	-	-	-
HGMR (L3)	0.16	1.01	1.84	0.18	0.73	1.28	0.19	0.39	0.59	0.09	0.28	0.55
FilterReg	0.15	1.15	1.92	0.31	0.50	0.93	0.34	0.32	**0.60**	0.42	0.11	**0.07**
FCGF (RANSAC)	0.11	0.81	2.89	0.70	0.45	1.84	0.87	0.34	1.38	**1.00**	0.19	0.74
D3Feat (RANSAC)	0.02	0.80	3.09	0.24	0.59	2.76	0.46	0.49	2.34	0.97	0.23	1.00
FPFH (RANSAC)	0.00	-	-	0.00	-	-	0.01	0.59	3.44	0.11	0.46	2.63
FCGF (TEASER++)	**0.38**	**0.48**	1.94	**0.86**	0.33	1.54	**0.95**	**0.28**	1.32	**1.00**	0.20	0.92
D3Feat (TEASER++)	0.23	0.62	1.98	0.65	0.56	1.79	0.79	0.51	1.63	0.97	0.40	1.09
FPFH (TEASER++)	0.00	-	-	0.00	-	-	0.01	1.18	3.38	0.30	0.93	2.54

For example, SR10 refers to the success rate measured give map size budget 10 $bytes/m^2$.

4.3 Evaluation

We evaluated the state-of-the-art point cloud registration methods using the proposed benchmark. Overall our benchmark successfully spotted interesting trade-offs between the map size and the success rate in real-world environments. Note that our benchmark is very challenging due to the map size constraints and the different data distribution between input LiDAR scans and the map. The detailed results are shown in Tab. 4.2 and 4.3. The success rate curves using different success thresholds are shown in Fig. 4.3.

Since data dimensions of different methods vary as shown in Tab. 4.1, we used the raw point format as the standard for feature point based methods. Let the number of raw points be N_r, x be the name of a method, F_x be the data dimension of method x. We computed the number of feature points N_x of method x by

$$N_x = \frac{3N_r}{F_x}. \tag{4.6}$$

Table 4.3: Results on the Argoverse Tracking Dataset. HGMR (L2) and HGMR (L3) outperformed all the other methods significantly when map size is small.

Metric name	**SR01**	**TE01**	**RE01**	**SR03**	**TE03**	**RE03**	**SR10**	**TE10**	**RE10**	**SR30**	**TE30**	**RE30**
map size($bytes/m^2$)		1			3			10			30	
ICP (pt2pt)	0.19	1.26	3.57	0.39	0.77	2.17	0.39	0.73	1.96	0.33	0.63	1.36
ICP (pt2pl)	0.03	1.20	3.29	0.29	1.18	3.10	0.33	1.03	2.62	0.43	0.72	1.85
GICP	0.09	1.10	3.40	0.29	0.93	2.36	0.34	0.78	2.07	0.42	0.43	1.07
Go-ICP	0.19	1.30	3.39	**0.62**	0.68	1.34	**0.67**	0.51	**1.02**	0.64	0.27	0.60
HGMR (L2)	**0.54**	0.74	1.70	0.40	**0.42**	**0.89**	-	-	-	-	-	-
HGMR (L3)	0.51	**0.67**	**1.64**	0.42	0.58	1.38	0.32	**0.47**	1.05	0.09	0.73	1.41
FilterReg	0.13	1.24	3.14	0.41	0.62	1.73	0.45	0.51	1.49	0.63	**0.15**	**0.33**
FCGF (RANSAC)	0.03	0.71	2.28	0.26	0.66	2.31	0.38	0.59	2.13	0.69	0.43	1.62
D3Feat (RANSAC)	0.01	0.79	1.76	0.11	0.59	2.05	0.21	0.55	1.99	0.80	0.38	1.45
FPFH (RANSAC)	0.00	-	-	0.00	-	-	0.00	-	-	0.03	0.57	3.10
FCGF (TEASER++)	0.03	0.93	2.97	0.24	0.96	2.66	0.36	0.93	2.51	0.69	0.82	2.08
D3Feat (TEASER++)	0.05	0.82	2.44	0.22	0.75	2.50	0.38	0.75	2.32	**0.86**	0.72	1.82
FPFH (TEASER++)	0.00	-	-	0.00	-	-	0.01	1.26	3.47	0.12	1.14	2.78

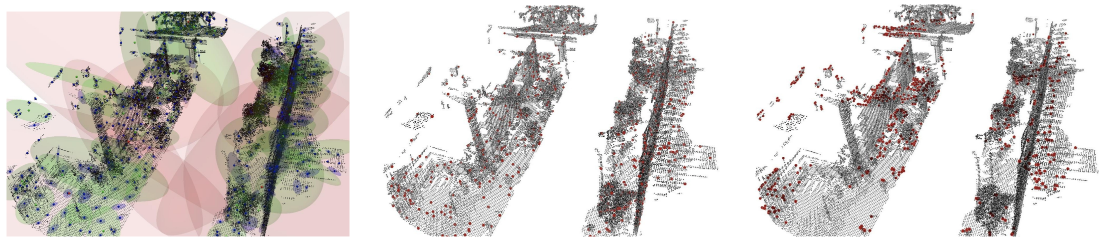


Figure 4.4: Visualization of different compressive map formats (a) hierarchical GMM tree from HGMR. The red, green, and blue colored ellipsoids represent a three-level GMM tree (b) Randomly downsampled points (red) (c) score-based downsampling used with D3Feat.

For example, corresponding to raw point based methods with 5000 raw points, a feature point based method with feature descriptor dimension 32 has a total dimension $32 + 3 = 35$, so the number of feature points corresponding to the 5000 raw points is approximately $\frac{3\times5000}{35} \approx 429$. For HGMR, constrained by the predefined tree structure, we evaluated with tree levels 2 and 3, and spanning node numbers $n \in [4, 6, 12, 16]$. This generated the different map size range of HGMR (L2) and HGMR (L3) against other methods in Fig. 4.1. See Tab. 4.4 for an intuitive data size comparison. Here we computed the raw data size despite the possibility of using additional compression tools, which can be applied on top of all the methods.

4.3.1 Raw Points

Among the methods using raw points for registration, we evaluated the classical point-to-point ICP, point-to-plane ICP, GICP, and the global method, Go-ICP. We compressed the map by randomly downsampling the raw points. Note that we computed the normals for GICP and point-to-plane on the raw dense map before downsampling to maintain precision. Overall, the success rate dropped as map size budget decreased. Go-ICP, as a global method, outperforms others by avoiding the local minima, but took at least seconds for a registration [157] thus is less practical. The global optimum found by Go-ICP was also not guaranteed to be the ground truth, especially for the Argoverse, since the point distributions were different in the source and the target point clouds.

4.3.2 GMMs

We evaluated the recent HGMR [43] and FilterReg [49] as the representatives of the GMM-based methods. We found that HGMR with a level-2 tree outperformed all the other evaluated methods significantly in the Argoverse when the map size is small. Given a fixed map size budget, the Gaussian model by HGMR is much smaller than the descriptors used by feature point based methods and thus allows more components as shown in Tab. 4.4. Therefore, it had a better spatial coverage than the feature point methods that used sparse feature point map. We also observed that increasing the tree size for HGMR did not lead to better registration results, as also shown in [43] for LiDAR datasets.

Table 4.4: Data size comparison and the corresponding average map sizes. The methods with higher data dimension can afford less N_x.

	unit	**dim.**	N_x		
Raw point	# points	3	100	1000	5000
Feature point	# point	35	9	86	429
GMMs	# weighted Gauss.	10	30	300	1500
KITTI	$bytes/m^2$	-	0.58	5.82	29.12
Argoverse	$bytes/m^2$	-	0.52	5.16	25.8

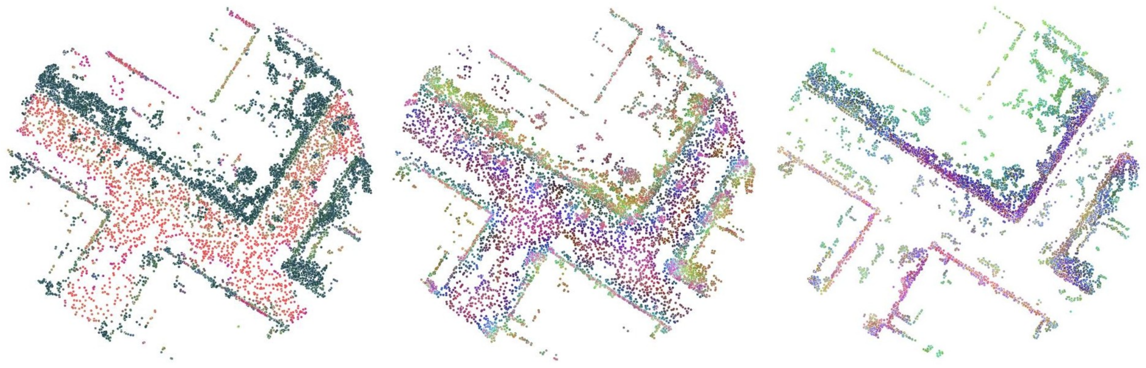
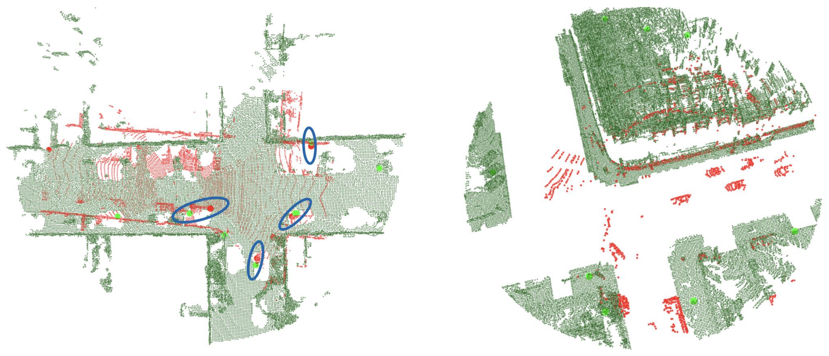

(a) FPFH (b) FCGF (c) D3Feat

Figure 4.5: Cropped maps of feature point based methods. The features were projected to three-dimensional space and visualized by the RGB colors. This visualization was downsampled to contain 10000 points so that the difference between random downsampling (FPFH and FCGF) and score-based downsampling (D3Feat) can be observed.

4.3.3 Feature Points

We considered feature point based methods together with robust registration algorithms, RANSAC and TEASER++ [156]. We first searched for the correspondence candidates by the descriptor matching and then filtered out noisy correspondences with RANSAC or TEASER++. In our experiments, the deep features, D3Feat [7] and FCGF [31], when used with either RANSAC or TEASER++, significantly outperformed the hand-crafted FPFH [112] in both datasets. For D3Feat, we downsampled the maps to fit the map size budgets by selecting points with higher learned scores. For FCGF and FPFH, we randomly downsampled the maps. A visualization of the downsampled maps for FPFH, FCGF, and D3Feat is shown in Fig. 4.5.

We observed that overall TEASER++ outperformed RANSAC. In addition, the success rates of FCGF and D3Feat in Argoverse were much worse than KITTI with smaller maps. As visualized in Fig. 4.6, the local cropped maps in KITTI overlap better with the LiDAR scans than in Argoverse, because the latter contains large regions that are invisible to the LiDAR scan. A severely downsampled feature point based map in Argoverse is more unlikely to overlap with the LiDAR scan and leads to lower success rate than the GMM-based method which covers more environment under the same map size budget.



(a) KITTI Odometry Dataset (b) Argoverse Tracking Dataset

Figure 4.6: An extremely downsampled case to show the differences between two datasets. The cropped maps are in dark green, the LiDAR scans are in red, and the downsampled maps are the light-green dots. (a) Many LiDAR scans overlap well with the cropped map and the FCGF (TEASER++) method can successfully find good correspondences (blue) in this case. The LiDAR scan is shown with the estimated pose here. (b) The LiDAR scan only overlaps by relatively smaller regions, and the downsampled cropped map does not overlap with the LiDAR scan well. This is common in Argoverse. The LiDAR scan is shown with the ground truth pose.

4.4 Discussion

In our experiments, we used a cropped map as the target and a LiDAR scan as the source. For the asymmetric methods, switching the source and the target might impact the results. Among the asymmetric methods we evaluated, we used the normals computed from the map for GICP and point-to-plane ICP because the map is much denser and less noisy. For HGMR, we build a GMM tree on the map because the compression ratio of GMM trees is more significant than downsampling the raw points. We also swapped the source and target for FilterReg and found it performing worse than the current configuration.

For the point based methods, we randomly downsampled the points if a score is not provided by the registration method. It is also possible to apply other downsampling techniques together with the evaluated methods to improve the performance. For example, Yin et al. [159] used point repeatability to prune the raw map for ICP.

Our results on feature point based methods suggest that, both the feature extraction and the correspondence filtering methods are crucial for the final performance.

TEASER++ [156] in general outperforms the classical RANSAC, but works best only when used together with the deep descriptors under our map size budget.

We also noticed that the deep learning based methods failed when map size is small mostly because the target map is feature point based and too sparse after intensive downsampling. A deep shape model based method might potentially increase the spatial coverage of the downsampled map and improve the performance. Further reducing the feature dimension without sacraficing the global feature matching accuracy is also worth more research.

4.5 Conclusion

In this benchmark, we consider single-frame LiDAR-to-map registration, which is an important module in a complex autonomous driving system. In real world conditions, autonomous vehicle systems include complicated interaction between multiple module and sensor modalities. Performance evaluation in this scenario remains an open research question. Our potential future works include the evaluation of multimodal registration algorithms such as registering an image to the LiDAR map.

Chapter 5

Image Registration to Compressed SfM Maps

In this chapter, we introduce our work on finding map representation for long-term image-to-SfM map (visual map) registration [27]. Here we focus on the problem of map sparsification: removing unimportant map parts to reduce map size without degradating the end task performance.

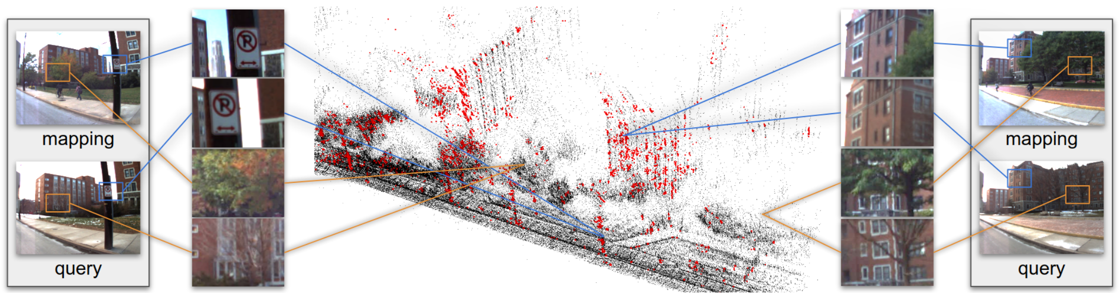


Figure 5.1: Given a map built from SfM, our proposed approach leverages GNNs and is able to identifies map points on stable structures (red points and blue squares), while discarding points that are prone to seasonal change, such as tree foliage (black points and orange squares).

5.1 Introduction

In long-term visual localization, a common strategy is to build and accumulate maps from the captured image streams, and then localize new incoming queries by matching against the accumulated map. In the presence of environmental changes, the accumulated map contains an increasing number of points and many of which are outdated. This will affect both the computational cost and the performance of localization in the long run. Therefore, the ability to identify and remove these invalid points is important for many applications that target dynamic environments, such as autonomous driving, field robotics, and Augmented Reality. Additionally, for devices with limited on-board memory, it enables keeping a compact map that only contains the most valuable information for future localization queries.

Existing works on map sparsification mostly fall into the category of subset selection, i.e., treating the 3D map as an over-sampled representation of a static world and aiming to select the most valuable point subset from them. The selection of point subset is typically formulated as a K-Cover problem. Assuming the map keyframes cover all the possible camera positions, the K-Cover algorithm encourages each keyframe in the map to observe K points under a total point number constraint [41, 84, 91, 101]. These methods are purely based on the historical data stored in the map, therefore lacking the ability to identify points invalidated due to environmental changes. When the environment changes, the map can only be updated by collecting new query data over the whole mapped area and solve the K-Cover problem again with the new query data, which is inefficient and expensive. Apart from sparsifying a 3D map, there are some works on selecting 2D key points, e.g., by predicting the persistency [42] or the repeatability [37] of visual features. However, the predictors proposed only take instantaneous measurements (such as local image patches) and not exploit the full context stored in the accumulated map.

Graph Neural Networks (GNN) have shown promising results on data with different structures, such as citation graphs [139], local feature matching [115] and visibility graphs [125]. In this work, we exploit this flexibility of GNNs to formulate map sparsification as a learning problem and overcome the limitations of previous methods. First, by modeling the SfM map as a graph, we can directly employ the context-rich SfM map as the GNN input in addition to instantaneous measurements. Second, in contrast to the K-Cover based methods that requires full-extent new queries to update the map, we are able to train a GNN with only partial queries and use it to sparsify the whole map. A main improvement from previous methods is the ability to incorporate the partial new data and select important points from the whole map according to the partial new data, as there is no trivial way for the baseline methods to do this without collecting new data that covers the whole mapped area.

To this end, we propose the first work that extracts features from SfM maps with a heterogeneous GNN. We first represent the SfM map with a heterogeneous graph, where 3D points, 2D key points and images are modeled as graph nodes, and the context such as the visibility between 2D and 3D points are modeled as graph edges. Afterwards, we use a heterogeneous GNN to predict map point importance scores based on the local appearance and the spatial context in the map graph. In addition, we propose two novel losses to guide the training: 1) a data-fitting term that selects points based on the appearance and the spatial distribution of the training query data, and 2) a K-Cover loss term that drives to sparse point selection with full-map coverage. When evaluated on an outdoor long-term dataset with significant environmental changes (Extended CMU Seasons [118]), our approach can select map points on stable and widely-visible structures (e.g., buildings/utility poles), while discarding points on changing object (e.g., foliage) or with highly repetitive texture (e.g., pavement). Compared with the K-Cover baseline [84], our approach outperforms in visual localization performance with the same map size.

5.2 Method

Given an SfM map and a set of localization queries recorded at different times in a large-scale dynamic environment, our goal is to select a subset of 3D map points that are most informative, i.e. result in high localization performance. To achieve this, we first turn the input SfM map into a heterogeneous graph (Sec. 5.2.1) and train an attention-based GNN (Sec. 5.2.2, 5.2.3) to predict the importance scores for 3D map points, which are then used to sparsify the map. Finally, we localize the testing query set against the sparsified map, and report the localization performance (Sec. 5.3). An illustration of our overall system flow is shown in Fig. 5.2.

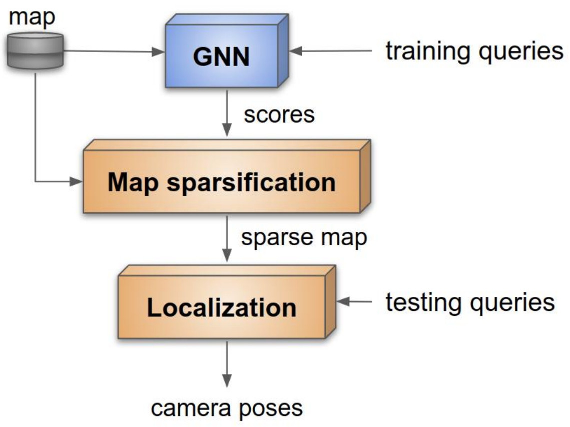


Figure 5.2: Overall framework. The proposed GNN learns to predict a score for each 3D point in the map. The predicted scores are used to sparsify the map. We report the performance of localizing a set of testing queries to the sparsified map.

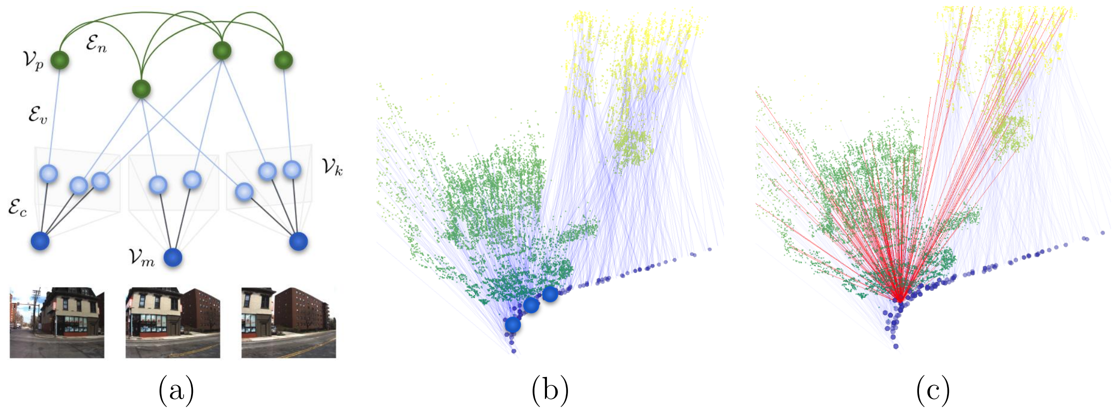


Figure 5.3: An SfM map as a heterogeneous graph and the network structure. a) A simplified graph: dark blue circles are image nodes $\mathcal{V}_m$, light blue circles are key point nodes $\mathcal{V}_k$, and green circles are 3D point nodes $\mathcal{V}_p$. The edges $\mathcal{E}_c$, $\mathcal{E}_v$, and $\mathcal{E}_n$ are containing edges, visibility edges, and kNN edges, represented by black, light blue, and green colors. (b) A real snapshot of the Extended CMU Seasons dataset. Image nodes $\mathcal{V}_m$ and visibility edges $\mathcal{E}_v$ are as blue dots and lines. The key point nodes $\mathcal{V}_k$ are not shown. The color on the 3D points $\mathcal{V}_p$ encodes the distance to the current query image with green being low values and yellow high values. Three image node positions corresponding to the images in (a) are labeled with with dark blue circles. (c) In each training iteration, we sample an image node and trace the corresponding edges to extract a subgraph to run our GNN. The $\mathcal{E}_v$ used to extract this subgraph are shown as red lines.

5.2.1 SfM Map as Heterogeneous Graph

A heterogeneous graph by definition is a graph structure that contains different types of nodes or edges. To represent an SfM map, three types of nodes are defined: 3D point nodes $\mathcal{V}_p$, 2D key point nodes $\mathcal{V}_k$, and image nodes $\mathcal{V}_m$. We also define three types of edges: visibility edges $\mathcal{E}_v$ connecting corresponding $\mathcal{V}_p$ and $\mathcal{V}_k$, kNN edges $\mathcal{E}_n$ connecting each $\mathcal{V}_p$ and its k nearest neighboring $\mathcal{V}_p$, and containing edges $\mathcal{E}_c$ connecting each $\mathcal{V}_k$ to the corresponding image $\mathcal{V}_m$. Each $\mathcal{V}_p$ might be connected to multiple $\mathcal{E}_v$s and $\mathcal{V}_k$s because it is observed by multiple mapping images. The SfM map is then represented with a heterogeneous graph $\mathcal{G} = \{\mathcal{V}_p, \mathcal{V}_k, \mathcal{V}_m, \mathcal{E}_v, \mathcal{E}_n, \mathcal{E}_c\}$. An illustration of our map graph is shown in Fig. 5.3(a)(b).

The per-point importance score is predicted based on local appearance and spatial context. We design our map graph to provide the information: first, the local appearance data are stored in $\mathcal{V}_k$ by embedding the key point descriptors extracted at the map building stage. Second, the spatial context is captured in kNN edges $\mathcal{E}_n$, which are derived from the 3D point positions stored in $\mathcal{V}_p$. The image nodes $\mathcal{V}_m$ do not carry features, but are used to trace connected $\mathcal{V}_k$ and $\mathcal{V}_p$ for ensuring the GNN selects enough number of $\mathcal{V}_p$ in the field-of-view of each $\mathcal{V}_m$, as shown in Fig. 5.3(c).

In practice, we store two sets of $\mathcal{V}_k$, $\mathcal{V}_m$, $\mathcal{E}_v$, and $\mathcal{E}_c$ in the map graph: one set is from the map and the other set is from localizing the query set on the map before sparsification. The first set is fed to the proposed GNN to provide information for score prediction. The second set is only available in the training area, and was only used to generate the point selection labels L_{gt} stored in $\mathcal{V}_p$ (Sec. 5.2.4).

Note that all the graph edges described above are directional. To be specific, $\mathcal{E}_n^{ji}$ represents a kNN edge from a neighbor $\mathcal{V}_p^j$ to the $\mathcal{V}_p^i$, and $\mathcal{E}_v^{wi}$ shows a visibility edge from key point $\mathcal{V}_k^w$ to map point $\mathcal{V}_p^i$, where i, j, w are node indices. The directionality of edges is useful in retrieving local subgraphs during network training (Sec. 5.2.3).

5.2.2 Graph Attention Network

To extract the spatial context from the map, we propose to aggregate the features from locally connected 3D point nodes with a Graph Attention Network (GATConv) [139, 142]. For a 3D point node $\mathcal{V}_p^i$, a GATConv layer is applied to fuse the input node

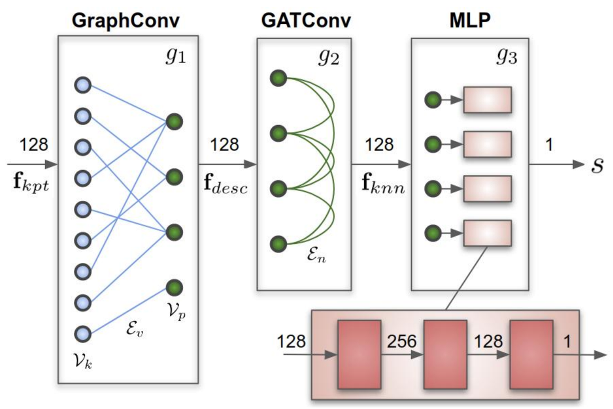


Figure 5.4: Our network takes the key point descriptors $\mathbf{f}_{kpt}$ and predicts a score s for each map point. We define three network layers: g_1 that aggregates descriptors to 3D points, g_2 that collects 3D local information, and g_3 as the final per-point MLP (pink blocks). A dark pink block is an MLP layer, which contains a linear layer and a LeakyReLU activation. The numbers above the arrows are feature dimensions

features and predict an output node feature. Formally, the GATConv operation is:

$$\begin{aligned} \alpha_{ij}^h &= softmax_j(a(\mathbf{W}^h\mathbf{h}_i, \mathbf{W}^h\mathbf{h}_j)) \\ \mathbf{h}_i^+ &= \sum_{h=1}^{H} \sum_{j\in\{1,\dots,k+1\}} \alpha_{ij}^h \mathbf{W}^h \mathbf{h}_j, \end{aligned} \tag{5.1}$$

where $\mathbf{h}_j \in \mathbb{R}^F$ is an input feature from $\mathcal{V}_p^j$ to node $\mathcal{V}_p^i$ with feature dimension F. The input features are from the $\mathcal{V}_p^i$ itself and the kNN nodes, where $j \in \{1, 2, \dots, k, i\}$ and k is the number of kNN nodes. The $\mathbf{W}^h \in \mathbb{R}^{F^+ \times F}$ is a shared weight matrix, α_{ij}^h is the normalized attention coefficient, H is the number of attention heads, $a(.) : \mathbb{R}^{F^+} \times \mathbb{R}^{F^+} \to \mathbb{R}$ computes the attention coefficients. We aggregate the multi-head GATConv outputs by simple summation. The output $\mathbf{h}_i^+ \in \mathbb{R}^{F^+}$ is the output feature with dimension F^+ stored on $\mathcal{V}_p^i$. Empirically, we found this GATConv outperformed GraphConv [68] and SAGEConv [58] for our application.

5.2.3 Heterogeneous Graph Neural Network

We design a heterogeneous GNN to extract features and perform score prediction from the aforementioned map graph. The motivation is that the key point descriptors, although not raw pixel values, still contain valuable appearance information, enabling us to infer the 3D point scores from the connected 2D key point descriptors. The heterogeneity here enables us to define different operations according to the node and edge types.

Our GNN comprises three stages: 1) a descriptor gathering layer g_1, 2) a local feature extraction layer g_2, and 3) a final Multilayer Perceptron (MLP) layer g_3. In g_1, we trace the connected $\mathcal{E}_v$ for each $\mathcal{V}_p$ to collect the connected key point descriptors stored in $\mathcal{V}_k$. The collected descriptors are sent to a Graph Convolutional layer (GraphConv) [68] with LeakyReLU activation and summation aggregation functions. The output of g_1 is an aggregated point feature $\mathbf{f}_{desc}$ carrying the local appearance information. In g_2, we use the GATConv layer (Sec. 5.2.2) to gather the nearby point features from the kNN $\mathcal{V}_p$, generating a local feature $\mathbf{f}_{knn}$ that captures spatial context. Finally, a 3-layer MLP g_3 is used to convert the point feature dimension to 1 and a sigmoid layer is used to constrain the predicted score value s to $[0, 1]$. The network structure is shown in Fig. 5.4.

Let $i, j \in \{1, 2, \ldots, N_p\}$ denote the map point indices and $w \in \{1, 2, \ldots, N_k\}$ be a key point index, where N_p and N_k are the total number of map points and key points. Let $\mathcal{G}$ denote the map graph, the score prediction steps are:

$$\begin{aligned} \mathbf{f}^i_{desc} &= g_1(\{\mathcal{V}^w_k | \mathcal{E}^{wi}_v \in \mathcal{G}\}) \\ \mathbf{f}^i_{knn} &= g_2(\{\mathbf{f}^j_{desc} | \mathcal{E}^{ji}_n \in \mathcal{G}\}) \\ s^i &= Sigmoid(g_3(\mathbf{f}^i_{knn})), \end{aligned} \tag{5.2}$$

where $\mathbf{h}_i = \mathbf{f}^i_{desc}$ and $\mathbf{h}^+_i = \mathbf{f}^i_{knn}$ in Eq. 5.1.

To facilitate GNN training on large-scale graphs, we sample a $\mathcal{V}_m$ to extract a local subgraph for each training batch and only run our GNN on the local subgraph. Given a $\mathcal{V}_m$, we first extract the connected $\mathcal{V}_k$ by tracing $\mathcal{E}_c$. Afterwards we trace $\mathcal{E}_v$ and $\mathcal{E}_n$ to extract the corresponding $\mathcal{V}^i_p$ and its neighbors. Finally, we trace the $\mathcal{E}_v$ connecting to the neighboring $\mathcal{V}^j_p$ for computing the neighboring $\mathbf{f}^j_{desc}$.

5.2.4 Training Losses

Our losses promote high scores on points with two properties: first, the descriptor distribution of the selected points should *align with the descriptors that are useful for training query localization*. Second, the selected points should *cover all the possible viewing poses*, so that all the queries would observe a sufficient amount of points within the field-of-view. We propose a training loss with two terms:

Data Fitting Term. Since the ILP baseline performs well in a static environment [84], we use it as an oracle to generate point selection labels. We first localize the training queries on the map, collecting the 2D-3D matches between the training queries and the map, and run the ILP baseline [84] to obtain the point selection results, which is a binary vector L_{gt}. The ILP baseline in this setting, denoted as ILP (query), factors out the environmental changes and performs well (Fig. 5.7(a)), but cannot be achieved in the real world unless the training queries cover the whole mapped area. The data fitting term is then computed by comparing the predicted scores $\mathbf{S}$ and L_{gt} with a Binary Cross Entropy (BCE) loss $\mathcal{L}_{BCE}$:

$$\mathcal{L}_{BCE} = BCE(L_{gt}, \mathbf{S}). \tag{5.3}$$

For the maps we evaluated with, we found the computation of ILP formulation is tractable to process the whole map. It is also possible to use IQP [101] for label generation, but in practice IQP is computationally intractable to run on large-scale maps without additional graph partition steps. The potential effect of graph partition on localization performance is beyond the focus of this chapter.

K-Cover Term. Training the network with $\mathcal{L}_{BCE}$ alone would only encourage point selection that aligns with L_{gt} in the training set, but it does not guarantee map point coverage across the whole map. To compensate this, leverage transductive learning and additionally encourage the sum of all the scores of all the $\mathcal{V}_p$ connected to each $\mathcal{V}_m$ to be close to a predefined positive integer K, which indicates the number of 3D points each image should observe to support robust localization. Empirically, we observed that this setup converges faster during training than the case not penalizing the samples larger than K. Upon satisfying the K-Cover constraint, we also encourage the score sparsity to select fewer points with an L_1 norm loss. Letting l be the index of image node $\mathcal{V}_m$, we define ϕ_l as the set of map point indices that selects the set of $\mathcal{V}_p$ whose connected $\mathcal{V}_k$ is within $\mathcal{V}_m^l$ (as the red edges in Fig. 5.3(c)). The score prediction of $\mathcal{V}_p^i$ is denoted as s_i. The final K-Cover loss is:

$$\begin{aligned} \phi_l &= \{i|\mathcal{E}_c^{lw} \in \mathcal{G} \cap \mathcal{E}_v^{wi} \in \mathcal{G}\}, \\ \mathcal{L}_{KC} &= \sum_l |K - \sum_{i\in\phi_l} s_i| + \lambda||\mathbf{S}||_1. \end{aligned} \tag{5.4}$$

By adding both terms, we propose the final loss as:

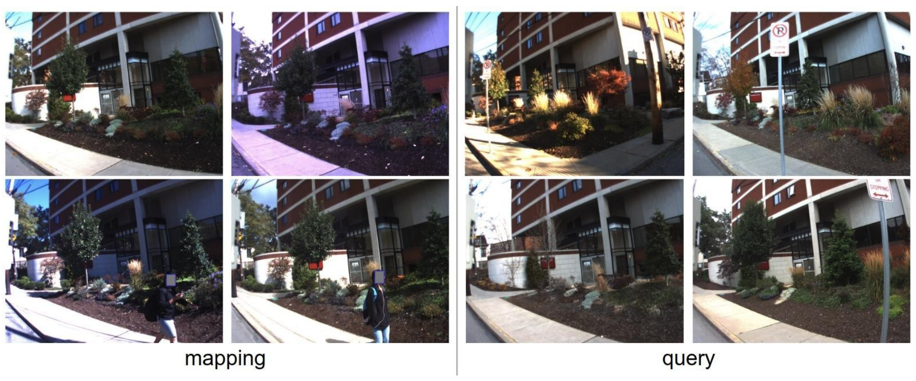
$$\mathcal{L} = \mathcal{L}_{BCE} + \mathcal{L}_{KC}. \tag{5.5}$$

The data split and usage is summarized in Tab. 5.1. Note that the training and testing queries are spatially non-overlapping and the pre-built map covers both the train and test areas. The role of the training queries is to provide up-to-date appearance information that cannot be obtained from the outdated map data, as we focus on the temporal appearance difference. In this case, the training and testing data should not overlap spatially but can overlap temporally.

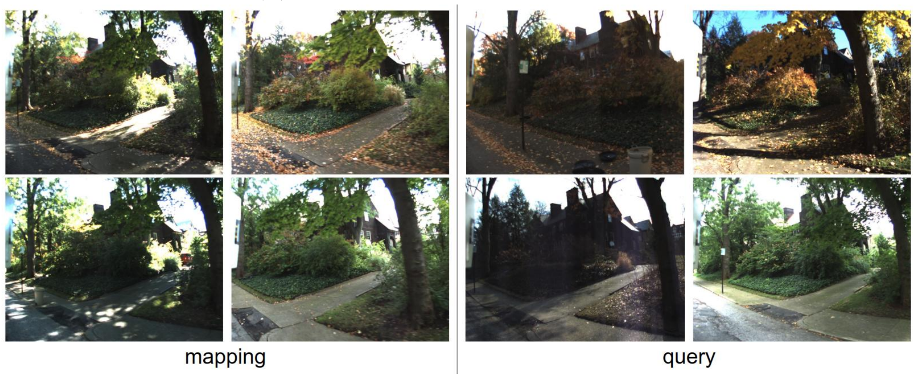
5.3 Evaluation

In this section, we describe the data preparation process, implementation details and experimental results.

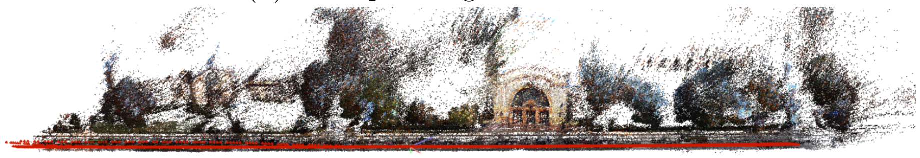
Data Preparation We evaluated our approach on Extended CMU Seasons dataset (license CC-BY-NC-SA 3.0) [6, 118], which consists of 12 sessions recorded by two cameras across months. To simulate the natural accumulation of map data, we used sessions 0-5 to build a multi-session map, and used sessions 6-11 as the query set to localize. The mapping and query sets have significantly different appearance. The map was built with Kapture [63]. The localization performance was measured by registering the query sets on the multi-session maps built from session 0-5 . We used 13 slices (scenes) for evaluation, including the Urban and Suburban slices (3-4, 6-16), and discarded the Park slices and slice 2, 5 due to poor localization performance on the raw multi-session map before sparsification. The 13 slices for evaluation contained various objects such as vegetation, buildings, and moving objects, and



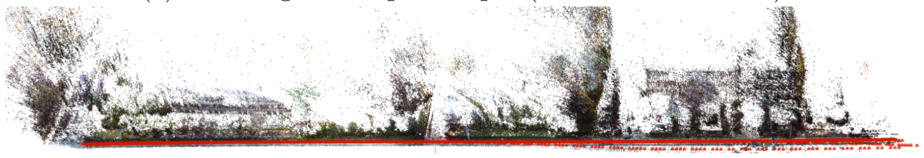
(a) Example images from slice 3



(b) Example images from slice 11



(c) Training set map example (camera 0 of slice 4)



(d) Test set map example (camera 1 of slice 4)

Figure 5.5: Example images from Extended CMU Seasons dataset. We observed large seasonal changes across the whole dataset. In (a)(b), on the left are the mapping image examples and on the right are query image examples recorded at similar locations. In addition, the Extended CMU Seasons dataset was recorded by two cameras. We used camera 0 (c) for training and camera 1 (d) for validation/testing. The training and test sets capture two sides of the road with no spatial overlap. The red dots at the bottom are the mapping image locations.

Table 5.1: The data splits by type and usage. There are two cameras in the Extended CMU Seasons dataset, noted by c0 and c1. We separated the 12 sessions temporally and used the old sessions (0-5) for mapping, the new sessions as queries (6-11).

Data Type	**Spatial**		**Temporal**		**Used for**
	train	test	old	new	
map ($\mathcal{G}$)	✓(c0)	✓(c1)	✓		$\mathcal{L}_{KC}$, $\mathcal{L}_{BCE}$
query (train)	✓(c0)			✓	$\mathcal{L}_{BCE}$
query (test)		✓(c1)		✓	not used

multiple weathers like sunny, cloudy, and snowy. An example of seasonal appearance changes is shown in Fig. 5.5(a)(b).

We further split the data from the two cameras (camera 0, camera 1), and used camera 0 of all the 13 slices for training, the camera 1 of slice 3 for validation, and the camera 1 of the other 12 slices for testing. The number of mapping/query images in each data set split are 17837/16077 for training, 1333/1428 for validation, and 16498/15627 for testing. Note that the camera 0 and camera 1 point towards two sides of the road and have no overlap as Fig. 5.5(c)(d).

Implementation Details The proposed GNN is implemented with PyTorch and Deep Graph Library (DGL) [142]. During the training process, we loop through the mapping image nodes $\mathcal{V}_m$ in the training set to extract subgraphs to run GNN on. A four-layer DGL node sampler ($\mathcal{V}_m \leftarrow \mathcal{V}_k \leftarrow \mathcal{V}_p \leftarrow\ knn\ \mathcal{V}_p \leftarrow \mathcal{V}_k\ of\ knn\ \mathcal{V}_p$) was used to extract the subgraph in each training iteration to provide necessary information. It took about 3.97s to process a map graph in average (with average 4.12×10^5 map points) on an Nvidia Quadro RTX 3000 GPU and an i7-10850H CPU @ 2.70GHz.

As for parameters, we used $k = 9$ to build kNN edges among 3D points, $K = 30$ and $\lambda = 0.01$ in the K-Cover loss. The ILPs [84] were implemented using Gurobi[57], and is configured with $b = 30$. We used $n_{desired} = 500$ to generate L_{gt}. The network was trained with an AdamW optimizer with learning rate 0.001 and βs $(0.9, 0.999)$ for 20 epochs. For each compared case, we selected the epoch with the best validation performance for testing.

Final evaluation is conducted with the Kapture localization pipeline [63]. Given a query image, it first retrieves the mapping images with similar global features, and then performs 2D-2D key point descriptor matching between the query image and the retrieved mapping images. The 3D points corresponding to the matched map key points are used for PnP with the matched query key points. The Kapture's R2D2 [108] descriptor is used in map building, localization, and as our input $\mathbf{f}_{kpt}$.

5.3.1 Localization Performance on Sparsified Maps

We compared the localization performance of our proposed heterogeneous GNN against a set of baselines. For each map sparsification method, we first obtained its point selection result, and reconstructed the multi-session map in Kapture format with only the the key points and descriptors that correspond to the selected points. We used the number of point descriptors remaining in the map (#kpts) as map size proxy, since these high-dimension descriptors (e.g., 128 for R2D2) occupied most of the map storage space. Three baselines were compared:

- **Random** : randomly select a subset of map points up to the allowed budget.
- **ILP (map)** : the conventional ILP [84], which assembles the K-Cover problem with 1) the visibility edges stored in the map, and 2) the per-point weight based on number of observations in the map.
- **ILP (query)** : the ideal ILP [84] that has access to test queries. The K-Cover problem is constructed using visibility edges from localizing the test queries on the map before sparsification, and points are weighted according to the number of observations during the test query localization. This approach indicates the ideal performance of ILP approach without environmental changes and cannot be achieved in the real world.

We obtained data points by sweeping the desired total point number $n_{desired}$[84]. For our method, we randomly selected points with predicted scores larger than 0.1. If there were not enough points with scores larger than 0.1 to satisfy $n_{desired}$, we randomly selected from the rest of the points. We observed that predicted score distribution is close to binary (due to the L_1 norm sparsity loss) and the point selection result is not sensitive to the score threshold.

Overall, our proposed approach outperformed the ILP (map) baseline in all the testing slices by achieving higher localization recall (success rate) under the same map sizes, as shown in Tab. 5.2 and Fig. 5.7. Qualitatively, we observed that compared

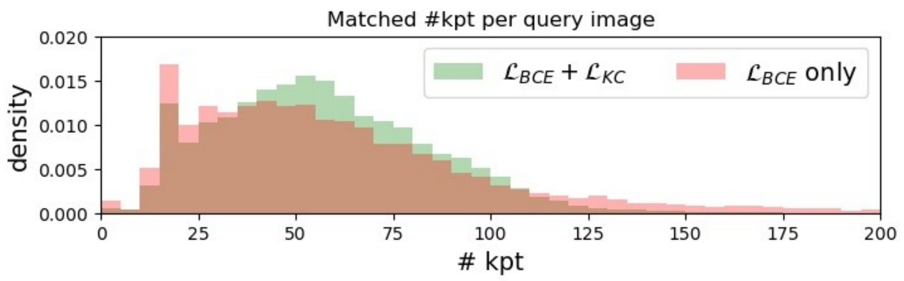
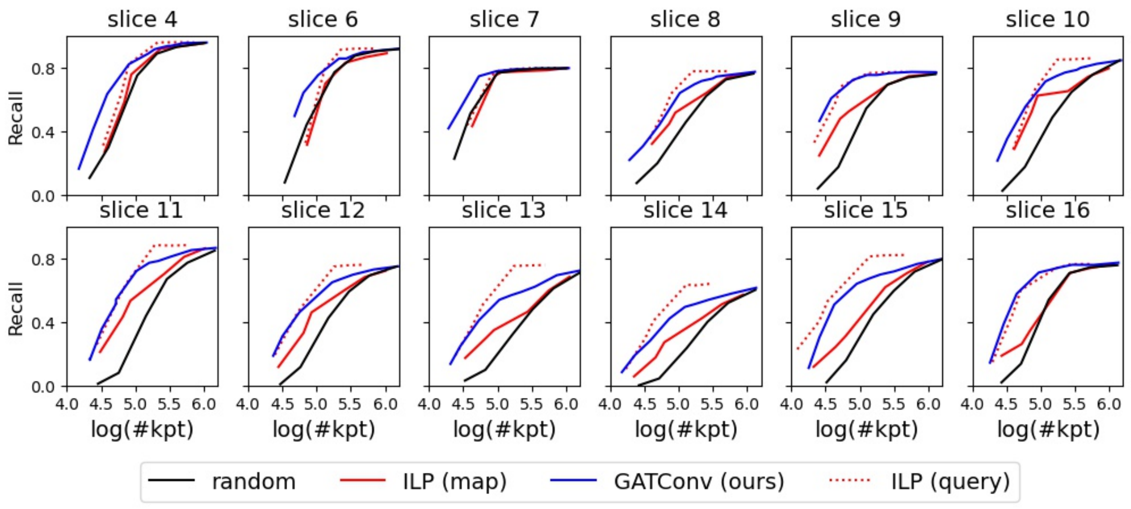


Figure 5.6: The density histogram of 2D-3D matching number for each testing query image during localization. After applying $\mathcal{L}_{KC}$ we observed less images with extreme number of matches, which is preferred for consistent localization performance under a map size budget. Both histograms are generated under the same budget (total #kpt from the 13 slices is $\sim 6.3 \times 10^5$).

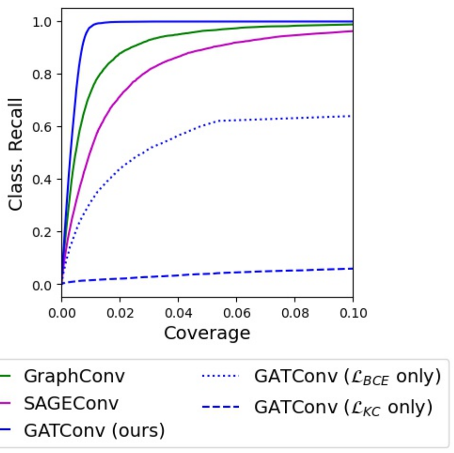
with the ILP (map) baseline, the proposed method selects map points on static structures that are more useful for query set localization, as in Fig. 5.8 and Fig. 5.9.

Network structures. We also compared the following configurations for the g_2 GNN layer: GraphConv [68], SAGEConv (with mean aggregation function) [58], and GATConv (with $H = 4$) [139]. The compared networks had the same feature dimensions and the LeakyReLU ($slope = 0.1$) activations. Our results showed GATConv outperformed GraphConv and SAGEConv significantly in terms of not only localization recall (Tab. 5.2) under the same map sizes, but also classification performance with respect to ILP (query) as shown in Fig. 5.7(b).

Training losses. Finally, the network trained without either $\mathcal{L}_{BCE}$ or $\mathcal{L}_{KC}$ performed worse than the one with combined loss, as shown in Tab. 5.2 and Fig. 5.7(b). The $\mathcal{L}_{BCE}$ was only trained in the training area, since no labels are available in the testing area. The $\mathcal{L}_{KC}$ were trained with the whole input map graph (which covers both the training and testing areas). Interestingly, although the $\mathcal{L}_{BCE}$ only configuration got the lowest training $\mathcal{L}_{BCE}$, adding $\mathcal{L}_{KC}$ improved the classification performance in the test set. We further observed that when localizing testing queries, the map sparsified with $\mathcal{L}_{KC}$ obtained less extreme numbers of matched key points (Fig. 5.6). This is favorable because each query obtained enough matches, but not too many that caused a waste in map storage.



(a) The recall vs. map size curves for each slice in the test set



(b) Classification performance

Figure 5.7: Localization and classification recall comparisons. (a) Our approach outperformed the ILP (map) and the random baselines in all test slices, achieving higher recalls (success rate) under the same map size budgets. On the other hand, the ILP (query) also significantly outperformed ILP (map), showing the impact of environmental changes on baselines. The recall error thresholds here are 0.25m and 2.0°. (b) Compared with ILP (query), the GATConv trained with the full proposed loss achieved the highest classification recall (ratio of selected positive labels) under the same coverage (ratio of the number of selected points against total number of points).

Table 5.2: Average recall under different map sizes. For each slice (a sequence in The Extended CMU Seasons dataset), we linearly interpolated the recall curves to obtain the recall numbers under the same number of key point descriptors, and computed the average recalls with respect to the number of images. Three recall thresholds were compared. The recall number represents the ratio of image samples with localization pose errors less than the corresponding recall threshold. As a reference, the average key point number before sparsification is $\sim 2.8 \times 10^6$.

Recall threshold	**0.25m, 2.0°**				**0.5m, 5.0°**				**5.0m, 10.0°**			
Avg. map size (10^4 #kpts)	3	5	10	20	3	5	10	20	3	5	10	20
Random	0.07	0.18	0.41	0.59	0.07	0.20	0.44	0.63	0.09	0.23	0.49	0.70
ILP (map)	0.15	0.31	0.53	0.64	0.19	0.36	0.59	0.69	0.25	0.43	0.66	0.76
GraphConv	0.31	0.48	0.64	**0.73**	0.34	0.52	0.69	0.77	0.39	0.58	0.76	0.85
SAGEConv	0.27	0.42	0.58	0.68	0.30	0.46	0.62	0.72	0.34	0.51	0.68	0.79
GATConv (ours)	**0.35**	**0.52**	**0.67**	**0.73**	**0.40**	**0.57**	**0.72**	**0.78**	**0.46**	**0.64**	**0.80**	**0.86**
GATConv ($\mathcal{L}_{BCE}$ only)	0.25	0.38	0.53	0.65	0.28	0.42	0.57	0.70	0.32	0.47	0.64	0.77
GATConv ($\mathcal{L}_{KC}$ only)	0.09	0.23	0.42	0.60	0.10	0.25	0.45	0.64	0.12	0.29	0.52	0.71
ILP (query)	0.24	0.46	0.69	0.80	0.30	0.53	0.75	0.85	0.38	0.60	0.83	0.92

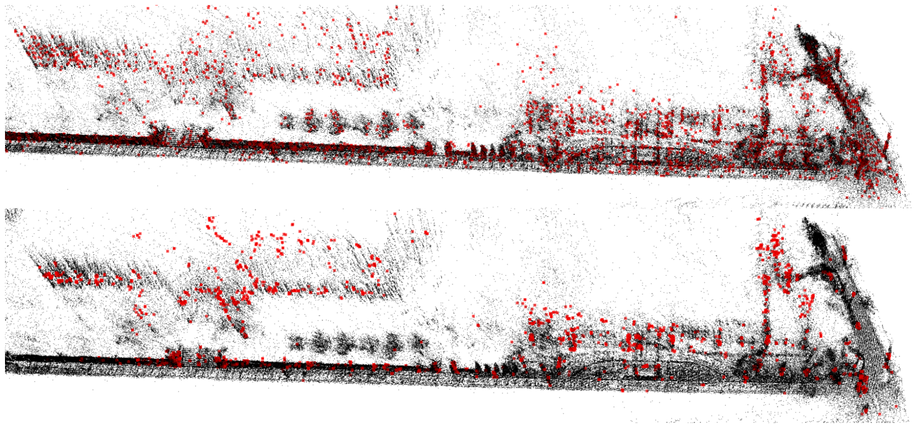
5.4 Discussion and Limitations

The heterogeneous graph used in this work is so flexible that it is easy to include more information as additional node or edge features. This implies a great potential for future works. Choices of additional information include timestamps (for capturing periodic environmental change) or the data from other sensors. It is also easy to apply other training losses to sparsify the map for different tasks other than conventional localization. Furthermore, we observed that certain objects, such as buildings and utility poles. are more likely to get higher scores. This implies the possibility of using semantic labels to assist point score prediction. It is also worth mentioning that the heterogeneous GNN framework can potentially be applied to other practical graphs, such as the factor graph for in SLAM. Comparing the GNN-based method with the existing factor graph sparsification works [13] is another interesting future direction. On the other hand, one important factor affecting the result is the point sampling strategy. Given the same set of predicted scores, different point selection strategies would lead to different performance. In our system, we used simple random down-sampling and a score threshold that achieved outstanding performance, but exploring different point sampling strategies can be an interesting future work.

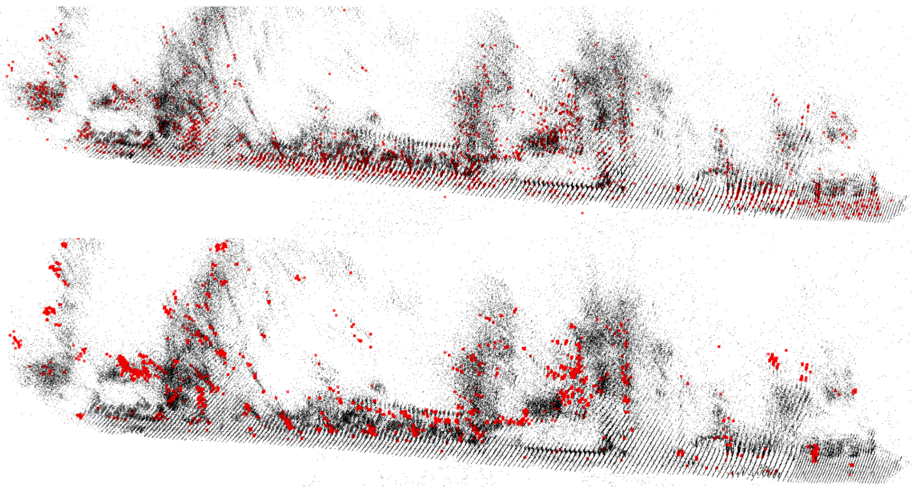
As for limitations, typically the key in map sparsification is to compress a map of a given scene, thus the generalization to an unseen scene has not been our focus. For the K-Cover setup to work, the camera trajectories at query time should be a subset of the camera trajectories in the map. This applies to ours and the related works. Besides, we only focused on removing points from an existing map, so the result is limited by the localization performance on the raw map. How to add/merge new information to the map is also worth exploring in the future. Finally, naive data splits (by camera and by slice) is used in our experiments, but in practice it is better to minimize the training set size to reduce the map update workload.

5.5 Conclusion

In conclusion, we proposed a heterogeneous GNN for visual map sparsification and proved its effectiveness in real-world environment. This work opens a new avenue for applying the abundant GNN related techniques to SfM applications.



(a) slice 3



(b) slice 11

Figure 5.8: Large-scale point selection results. The upper row is the results from ILP (map) and the lower row is ours with a 0.1 score threshold. The black points are the map 3D points before sparsification and the red points are the selected points. Our method selects points on static structures, such as building walls, utility poles, and tree stems and avoids foliage that changes across seasons.

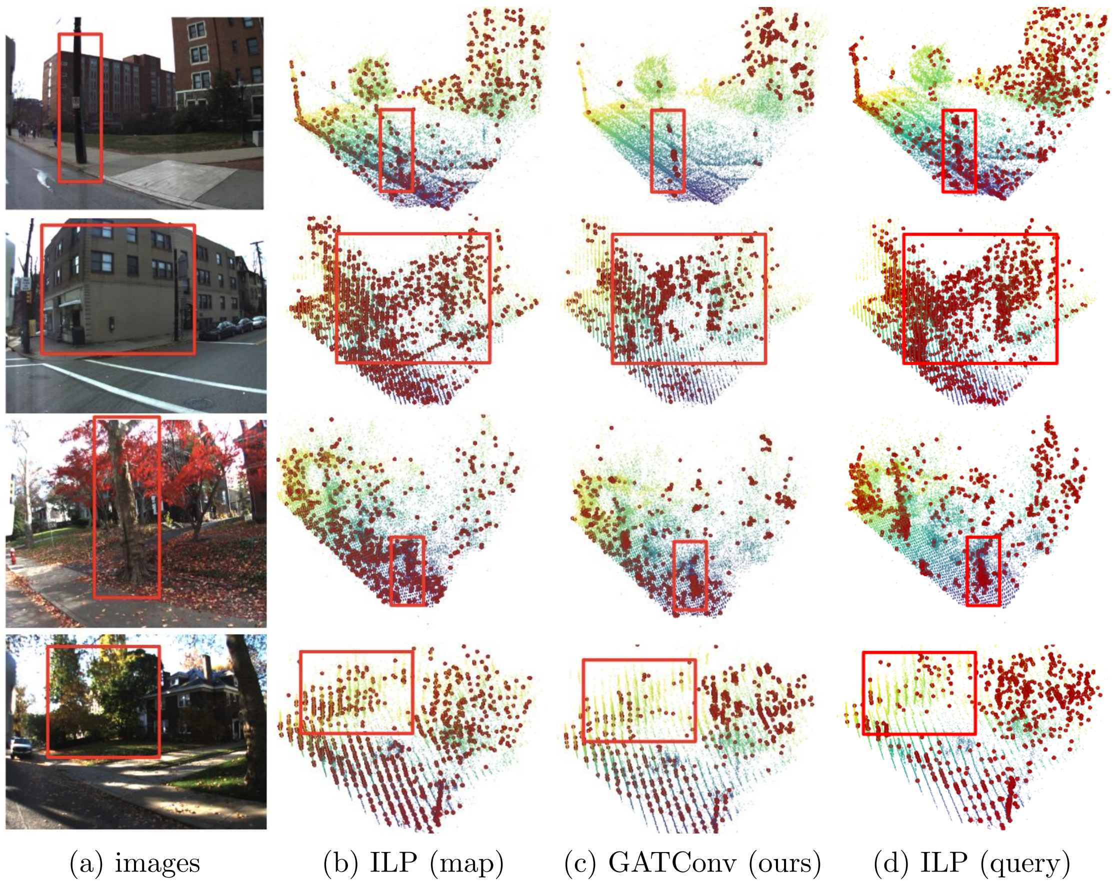


Figure 5.9: Qualitative visualizations. The camera positions are at the bottoms of the point cloud visualizations (b)(c)(d). The corresponding parts in each row are labels by red boxes. Overall, we observe that the point selection of ILP (map) is less discriminative in selecting static points than ILP (query) and ours. We compared the cases with similar numbers of key points so the total 3D point number varies.

Chapter 6

Neural Radiance Field with LiDAR maps

In this chapter, we will focus on exploring neural representation as a novel map format. Since published in 2020, 3D implicit neural representation such as Neural Radiance Field (NeRF) [93], has gained major popularity in the computer vision community [48]. This powerful representation can encode the geometry and appearance of 3D objects or scenes in detail, and performs novel view synthesis (NVS) with high visual quality. More recently, it also attracts certain interests in robotics field, and has been applied to robotic applications such as path planning [3], localization [85], dense visual SLAM [110], and active mapping and planning [164] .

This brings up two important questions to answer: 1) how to build a NeRF in a practical setting? The majority of existing NeRF systems only handles small scenes because a NeRF [93] requires densely-sampled images to build, but here we are interested in large-scale outdoor scenes. We combine LiDAR sensors with images to overcome this challenge. 2) How to make the built neural useful? Considering the potential use as a realistic simulator, we explored data augmentation, object detection, and season change applications to demonstrate the rich future possibility of neural maps in the robotics field.

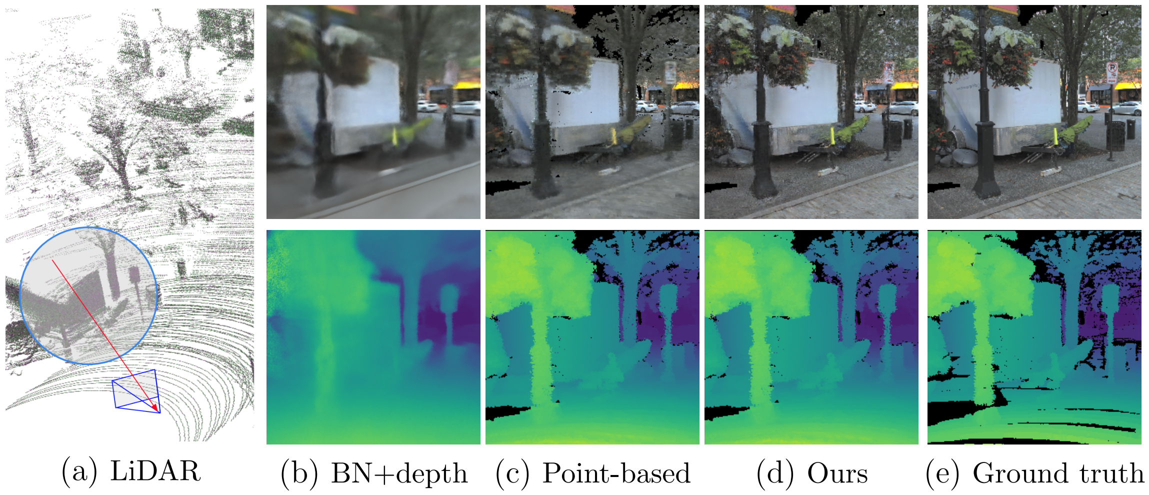


Figure 6.1: We design a novel view synthesis system from outdoor camera-LiDAR datasets with a point-based NeRF framework and 2D conditional GANs. (a) A LiDAR map (gray points) and queried novel view (blue). (b)-(e) Our method outperforms previous BlockNeRF (with LiDAR depth supervision) and point-based NeRF on Argoverse 2 dataset.

6.1 Introduction

Despite the fact that recent works have made massive improvements in novel view synthesis (NVS) for small scenes [9, 47, 76, 87, 93, 95, 102], large-scale outdoor scenes – such as street views and parks – are still challenging. Improving the NeRF results on large-scale outdoor scenes would greatly benefit multiple applications, such as realistic simulators for robot navigation [3], localization [85], active mapping and planning [164], and novel view augmentation [94].

The transition from indoor to outdoor presents a non-trivial challenge. Typically, the training of NeRF models demands densely sampled views to achieve good accuracy [34]. However, collecting densely sampled training views in outdoor scenarios requires much labor and storage space, and the camera trajectories in outdoor settings are typically biased (straight and along the lane). Many parts of the scene are often only observed by a limited number of views and range of view angles. This lack of data coverage issue of common outdoor datasets [45, 51, 150] is also addressed by previous works [107, 131], where specially collected datasets instead of common public outdoor datasets were used.

Furthermore, previous methods using simple MLPs to represent large blocks tend to generate blurry results (Fig. 5.1 (b)). On the other hand, it has been demonstrated that the demand of dense training views can be effectively reduced by geometry priors [34, 149], and modern outdoor robots – such as autonomous vehicles – are often equipped with LiDAR sensors in addition to cameras. In contrast to previous LiDAR-assisted works [18, 107] that used LiDAR scans only as supervision or as a guidance for ray sampling, our approach treats the LiDAR map as sparse samples of the environment and directly distributes localized embeddings on it.

Using localized embeddings instead of global representation can ease the burden of memorizing the whole scene with a single MLP in NeRF, leading to better embedding locality and convergence speed [59, 81, 154]. PointNeRF [154] embedded per-point features to 3D point clouds (from CNN prediction or COLMAP [121, 122]) and aggregated these point cloud embeddings along ray samples for volume rendering. However, the method in [154] is not directly applicable to the common camera-LiDAR datasets we target. The real-world LiDAR maps are prone to noise due to imperfect conditions such as bad weather. Simply using the point-based NeRF on outdoor LiDAR maps leads to noisy and unsatisfactory image quality, as in Fig. 5.1 (c). The 3D point cloud refinement techniques proposed in [154] requires the photometric constraints from the texture of dense training views, and thus are not suitable for the outdoor datasets where the views are sparser and the scenes are more complex. Instead of 3D point cloud refinement as [154], we propose to leverage strong 2D image refinement in this work.

The proposed pipeline takes a neural 3D point cloud (i.e. the LiDAR map with embeddings) as input and aggregates the LiDAR embeddings to perform volume rendering. In addition, we propose a tight sampling strategy in contrast to the naive radius-based counterpart in [154] to make samples better align with the LiDAR geometry prior. Finally, we refine the quality of synthesized views in 2D with a conditional GAN (cGAN) [65]. The proposed cGAN module can be trained end-to-end without additional data, and largely improves the final image quality. Besides common image metrics, we also show that the Detectron2 [153] detection results from our rendered images are closer to the results from ground truth images than the baselines. Last but not least, the proposed system can serve several interesting applications, including data augmentation for training a pose regression network [12] and seasonal appearance rendering.

In summary, our pipeline amalgamates the strengths of neural radiance field (for deep implicit 3D representation), LiDAR maps (for geometric priors), and cGAN (for deep realistic appearance rendering). The demonstrated applications also show foreseeable potential of our system to benefit tasks that require a richer neural map representation.

6.2 Method

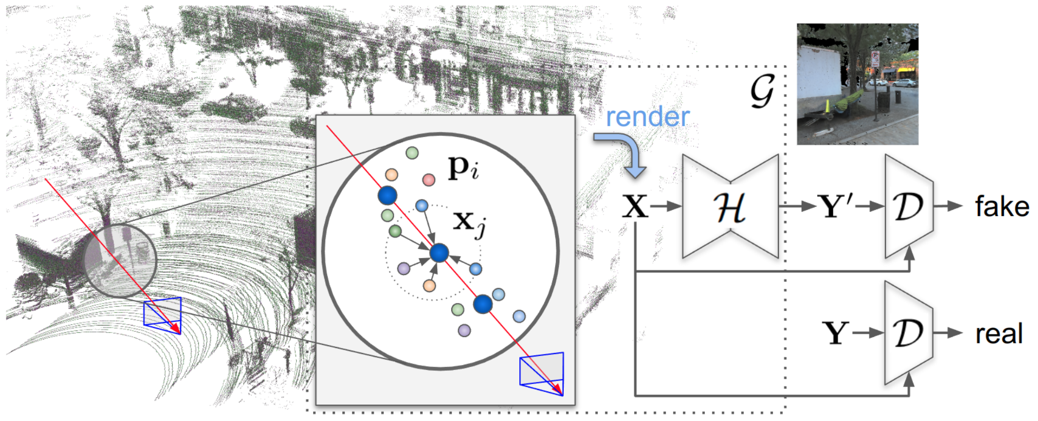
Given a camera-LiDAR data sequence with known poses, we first build a LiDAR map $\mathcal{P}$ by accumulating the LiDAR scans, and then assign trainable per-point embeddings to the LiDAR map as our localized neural radiance field representation (Fig. 6.2). These neural LiDAR points represent sparse samples of the underlying neural radiance field.

For a queried novel view pose, we perform volume rendering that interpolates and aggregates LiDAR point embeddings to generate pixel colors (Sec. 6.2.1). This volume rendering step generates an initial image $\mathbf{X}$, which is refined with a cGAN to generate a final image $\mathbf{Y}'$ (Sec. 6.2.2). We train the whole pipeline in an end-to-end fashion with supervision from training source views $\mathbf{Y}$, conditional adversarial losses (Sec. 6.2.2), and LiDAR map geometry (Sec. 6.2.3).

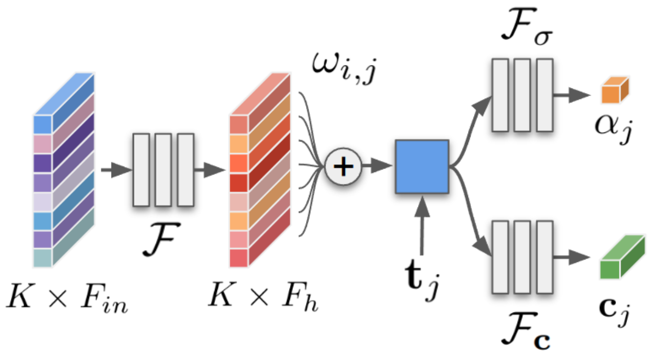
6.2.1 Point-based Volume Rendering

We follow the conventional volume rendering method and compute the per-pixel radiance via ray marching [93, 154]. First, a ray is drawn from the camera center to the pixel center in 3D space, and M ray sample positions $\{\mathbf{x}_j \in \mathbb{R}^3 | j = 1, \ldots, M\}$ are selected along the ray. Afterwards, the sample color $\mathbf{c}_j$, and density σ_j are computed at each of the M points. Finally, samples along the ray are accumulated to compute the pixel color $\mathbf{C}$:

$$\begin{aligned} \mathbf{C} &= \sum_{j=1}^{M} \tau_j (1 - \exp(-\sigma_j \delta_j)) \mathbf{c}_j, \\ \tau_j &= \exp(-\sum_{t=1}^{j-1} \sigma_t \delta_t), \end{aligned} \tag{6.1}$$



(a) System pipeline



(b) Network structure

Figure 6.2: (a) We perform spatial interpolation to aggregate the LiDAR map embeddings (smaller dots $\mathbf{p}_i$) onto volume rendering ray samples (larger blue dots $\mathbf{x}_j$). A cGAN is used to refine the volume rendering output $\mathbf{X}$, where the generator $\mathcal{G}$ contains volume rendering parameters and a CNN $\mathcal{H}$ that translates the volume rendering output image $\mathbf{X}$ to a refined image $\mathbf{Y}'$. The discriminator $\mathcal{D}$ aims to predict real or fake based on feeding the ground truth image $\mathbf{Y}$ or the generated image $\mathbf{Y}'$. (b) We process the per-LiDAR point information with an MLP $\mathcal{F}$ and aggregate the processed embeddings by spatial interpolation with weights $\omega_{i,j}$. View-based appearance embeddings $\mathbf{t}_j$ can be incorporated (blue block). Finally we predict sample color $\mathbf{c}_j$ and density α_j with the other two MLPs $\mathcal{F}_\sigma$ and $\mathcal{F}_\mathbf{c}$.

where δ_j and δ_t are the intervals between adjacent ray samples, and τ_j is the transmittance accumulated from the density of the ray samples between $\mathbf{x}_j$ and the camera center.

Point Sampling with Priors

A main challenge in large-scale volume rendering is to determine the positions of ray samples $\mathbf{x}_j$. Ideally, $\mathbf{x}_j$ should cover the potentially occupied regions, but in practical systems, the number of samples and their coverage is limited. The geometry prior from LiDAR maps provides important guidance for sampling at the occupied locations and skipping the large empty space.

Specifically, we first uniformly sample M_0 positions along a camera ray and compute the distances ρ from each sample to its nearest LiDAR point. Among the samples with $\rho < \zeta$, where ζ is an assigned radius threshold, we select the first M samples that are closest to the camera center. To collect local information from our point-based neural radiance field, we query the k-nearest LiDAR point neighbor set ϕ_j^{kNN} for the M selected ray samples $\mathbf{x}_j$:

$$\phi_j^{\mathrm{kNN}} = \{\mathbf{p}_i \in \mathbb{R}^3 | \; ||\mathbf{p}_i - \mathbf{x}_j|| < \zeta \text{ and } \mathbf{p}_i \in \mathcal{P}\}. \tag{6.2}$$

In practice, k varies among samples but is upper-bounded by an assigned parameter K. The radius threshold ζ depends on the noise and density of the LiDAR map.

The above radius-based selection simply dilates the 3D extent of LiDAR point distribution. However, for datasets with good LiDAR quality (such as autonomous driving datasets), the LiDAR points are usually tightly distributed near object boundary, and we only need to fill the holes without dilation. The dilated regions are most likely to be empty, and we would place unnecessary ray samples in those empty regions (Fig. 6.4 (b2)) if using naive radius based selection like [154].

To solve this issue, we propose to trust the LiDAR geometry more by tightening the extent of $\mathbf{x}_j$. This can be achieved by removing the $\mathbf{x}_j$s that are not surrounded by the LiDAR points in ϕ_j^{kNN} as in Fig. 6.4 (a1)(a2). Let $\mathbf{p}_0$ be the nearest LiDAR point in ϕ_j^{kNN}, we discard the sample $\mathbf{x}_j$ if

$$(\mathbf{x}_j - \mathbf{p}_0) \cdot (\mathbf{x}_j - \mathbf{p}_i) > 0, \; \forall i \in \{1, \ldots, k\}. \tag{6.3}$$

A visualization of the tightened ray samples is in Fig. 6.4 (b3). A quantitative comparison is in Tab 6.4.

Feature Aggregation

After collecting $\mathbf{p}_i \in \phi_j^{\text{kNN}}$, we aggregate their map embeddings to predict the color and density for sample $\mathbf{x}_j$. Specifically, we first concatenate the LiDAR point embeddings $\mathbf{f}_i \in \mathbb{R}^{F_m}$ with spatial offsets $\mathbf{o}_{i,j} = \mathbf{p}_i - \mathbf{x}_j$, and then pass the concatenated embeddings through a light-weight MLP $\mathcal{F}$ to obtain processed embeddings $\mathbf{f}'_{i,j}$. Afterwards, we aggregate $\mathbf{f}'_{i,j}$ onto $\mathbf{x}_j$ via spatial interpolation to obtain the sample embedding $\mathbf{h}_j \in \mathbb{R}^{F_h}$:

$$\begin{aligned} \mathbf{f}'_{i,j} &= \mathcal{F}(\text{concat}(\mathbf{f}_i, \gamma(\mathbf{o}_{i,j}))), \\ \mathbf{h}_j &= \frac{\sum_{i=1}^k \omega_{i,j}\mathbf{f}'_{i,j}}{\sum_{i=1}^k \omega_{i,j}}, \end{aligned} \tag{6.4}$$

where $\gamma(.)$ represents the positional encoding function. Using the spatial offset $\mathbf{o}_{i,j} \in \mathbb{R}^3$ instead of simple distance makes $\mathbf{f}'_{i,j}$ anisotropic and richer. The weights $\omega_{i,j}$ are designed to favor embeddings from closer LiDAR points:

$$\omega_{i,j} = \exp(-\beta||\mathbf{p}_i - \mathbf{x}_j||). \tag{6.5}$$

The choice of weighting function should depend on the LiDAR sensor and map characteristics. In our case, LiDAR points are locally dense around the object surface, and we found Eq. 6.5 works well in our experiments. Finally, the per-sample color and density, $\mathbf{c}_j$ and α_j, are predicted from $\mathbf{h}_j$ with two other MLPs, $\mathcal{F}_\alpha$ and $\mathcal{F}_\mathbf{c}$, as shown in Fig. 6.2 (b).

We also make Lambertian assumption and discard view direction dependency since modeling reflective surfaces is not our focus (see Sec. 6.5). Although one can also include view direction as in [131, 154] or model the lighting for specific objects with object shape priors as in [141], we found this setup suffice the applications we explored (Sec. 6.4).

Additional Appearance Embeddings

Additional latent variables can be incorporated to manipulate synthesized image appearance. Here we use view-based embeddings $\mathbf{t}_j$ (Fig. 6.2 (b)). A season change demonstration with timestamps as $\mathbf{t}_j$ is presented in Sec. 6.4.3.

6.2.2 Image Refinement with cGAN

Conditional Adversarial Networks (cGANs) have been known for the ability to perform image translation that generates visually pleasant details [59, 65, 168], and is trainable from small datasets [80, 128]. Overall, we follow the pix2pix framework [65]. For a training pair $(\mathbf{Y}, \theta)$ consisting of a training source view image and its pose, we perform volume rendering from the neural LiDAR map to generate an initial image $\mathbf{X}$ from θ, and use a CNN $\mathcal{H}$ (Fig. 6.2 (a) and Fig. 6.3) to translate $\mathbf{X}$ into a realistic domain image $\mathbf{Y}' = \mathcal{H}(\mathbf{X}, z)$, where z is the introduced noise vector described in [65]. We train the whole pipeline in an end-to-end fashion, so our generator $\mathcal{G}$ not only contains parameters from $\mathcal{H}$, but also the volume rendering MLPs, $\mathcal{F}, \mathcal{F}_\sigma, \mathcal{F}_\mathbf{c}$, and the neural map point embeddings $\mathbf{f}_i$ (Sec. 6.2.1). A discriminator $\mathcal{D}$ takes the concatenation of $\mathbf{X}$ and $\mathbf{Y}'$, $\mathbf{X}$ and $\mathbf{Y}$ as input, and predicts high scores for $\mathbf{Y}$ and low scores for $\mathbf{Y}'$. The cGAN loss contains an adversarial term and a $\mathcal{L}_1$ norm term:

$$\begin{aligned} \mathcal{L}_{\text{cGAN}} =& \mathbb{E}_{x,y}[\log \mathcal{D}(\mathbf{X}, \mathbf{Y})] + \mathbb{E}_{x,z}[\log(1 - \mathcal{D}(\mathbf{X}, \mathbf{Y}')], \\ \mathcal{L}_1 =& ||\mathbf{Y} - \mathbf{Y}'||_1. \end{aligned} \tag{6.6}$$

Finally, we solve the following optimization problem:

$$\arg\min_{\mathcal{G}} \max_{\mathcal{D}} \mathcal{L}_{\text{cGAN}}(\mathcal{G}, \mathcal{D}) + \lambda \mathcal{L}_1(\mathcal{G}). \tag{6.7}$$

In practice, we discard the noise vector z to obtain consistent image outputs for natural video rendering. The same strategy is also used in CycleGAN [168]. The application of CNN $\mathcal{H}$ also enables us to slightly dilate the RGB coverage on the cGAN output (see the difference in black area in Fig. 6.9 (d) and (e)). We use a small 6-layer autoencoder with three ResNet blocks [61] in the middle as $\mathcal{H}$. A relevant observation of advocating small receptive field for refinement CNNs is also mentioned by GANCraft [59], indicating that small receptive field is beneficial for generating

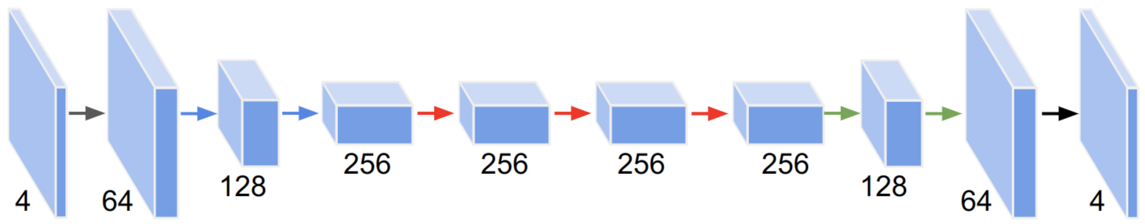


Figure 6.3: Our image refinement network $\mathcal{H}$. The downsampling 2D convolutional layers with stride 2 is represented by blue arrows, and the upsampling transposed 2D convolutional layers are represented by green arrows. The ResNet blocks in the middle are represented by red arrows. Two 2D convolutional layers (black arrows) with stride 1 are used at input and output. LeakyReLU activations are appended to each layer except for the output layer.

view-consistent results. Interestingly, we can further control the output image style of cGAN with the strength of $\mathcal{L}_{\text{cGAN}}$. Stronger $\mathcal{L}_{\text{cGAN}}$ adds more fine details to $\mathbf{Y}'$, leading to lower PSNR but higher LPIPS (Sec. 6.3.5). This system can be trained with only the log sequence ($\sim$155 images (Sec. 6.3)) without additional data.

6.2.3 LiDAR Depth Loss

Here we present an example to further demonstrate the importance of LiDAR depth loss, especially in the environment lacking photometric constraints. Given a camera frame t, we computed the photometric errors along the epipolar lines in its consecutive frames $t-1$ and $t+1$. In Fig. 6.5, we can observe the flat bottom in the photometric curves with respect to depth, showing there is no unique minimum to find the optimal depth in this condition.

We introduce this LiDAR depth loss to constrain the estimated depth to be close to the LiDAR measurements. To achieve this, we render pseudo ground truth depth from the LiDAR geometry as supervision. We assume the LiDAR measurements form a thin layer of opaque material on object surfaces to render the pseudo ground truth depth image, denoted as $\mathbf{D}^l$. Specifically, for each ray sample $\mathbf{x}_j$, if its distance to the closest LiDAR point is less than an assigned threshold μ, we consider it opaque (i.e. $\exp(-\sigma_j\delta_j) = 0$ in Eq. 6.1), otherwise transparent ($\sigma_j = 0$). Next, we accumulate the samples to obtain the pseudo ground truth depth image $\mathbf{D}^l$ with Eq. 6.1. Let $\mathbf{D}$ be

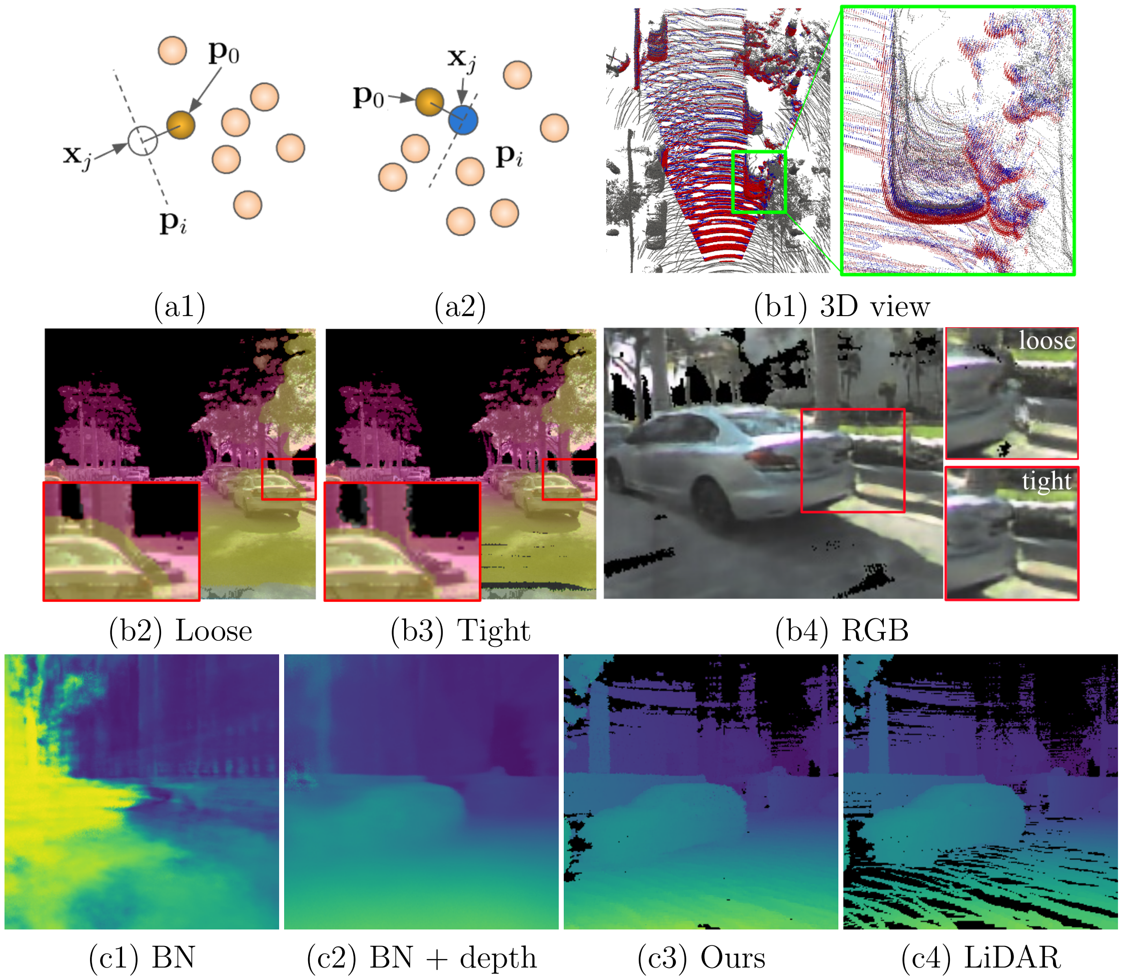


Figure 6.4: (a) Examples of the proposed tight sampling strategy. The LiDAR points $\mathbf{p}_i$ in ϕ_j^{kNN} are in light orange, and the nearest LiDAR point $\mathbf{p}_0$ is in dark orange. The white sample $\mathbf{x}_j$ in (a1) is discarded because all the $\mathbf{p}_i$s are in the same side as $\mathbf{p}_0$, meaning $\mathbf{x}_j$ is not surrounded by LiDAR points. The blue sample in (a2) remains because its surrounded by LiDAR points. (b1) The LiDAR map points are in gray, and the loose and tight ray samples are in red and blue. (b2)(b3) Overlaid initial depth projection and the rgb of loose and tight sampling strategies. (b4) Artifact around object boundary due to mismatch between depth and rgb (upper right). The object boundary from the tight sampling strategy (lower right) is better aligned with rgb. (c) The proposed depth loss pulls rendered depth in (c2)(c3) closer to the LiDAR depth (c4).

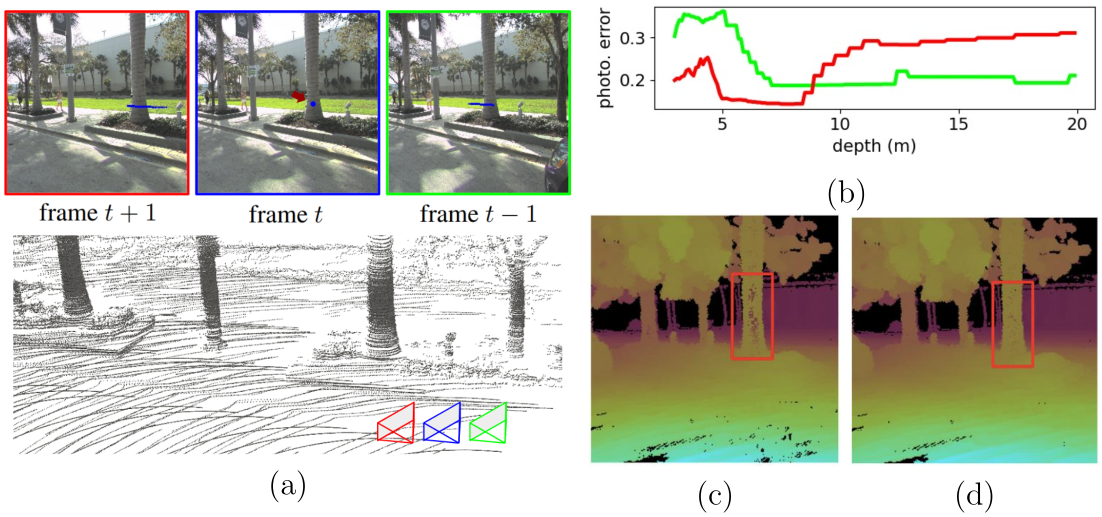


Figure 6.5: The effect of LiDAR depth loss. (a) The trunk region (blue dot on frame t) does not contain distinguished photometric information for determining depth. (b) The photometric error curves (within depth range $3 - 20m$) of the frame t blue dot in (a) on frame $t-1$ and frame $t+1$ are shown in green and red. The corresponding epipolar line segments are denoted by blue line segments on the frame $t-1$ and frame $t+1$ images in (a). Note that the photometric error curves in (b) contain large flat regions and no unique minimum for determining the optimal depth. The validation depths at this region w/o and w/ the LiDAR depth loss are shown in (c) and (d). We observed that the LiDAR depth provides guidance for correct depth on the trunk in (d).

the depth map output from Sec. 6.2.1, our LiDAR depth loss is:

$$\begin{aligned}\mathcal{L}_l &= \frac{1}{|\phi_{\text{valid}}|}\sum_{n\in\phi_{\text{valid}}} |D_n - D^l_n|,\\ \phi_{\text{valid}} &= \{n|\mathbf{D}^l_n > 0\},\end{aligned} \tag{6.8}$$

where ϕ_{valid} is the set of pixels with valid LiDAR depth.

A potential alternative way is to apply an additional loss term that enforces σ_j of the ray samples close to LiDAR points to be large. However, the LiDAR sensor only returns measurements on object surfaces. Ray samples inside objects might be far away from LiDAR points but still have high density, and thus their density cannot be constrained by the nearest distance to LiDAR points. In our case, the ray samples inside objects are not constrained because they are occluded by the opaque samples on the object surfaces (close to LiDAR points) along the same ray.

6.2.4 Moving Object Removal

Our focus is to render static scenes, and the neural LiDAR maps are also assumed static. The dynamic objects captured by training views and LiDAR scans would introduce unwanted inconsistent appearance information and cause blurry shadows in the rendered images. To overcome this issue, we use the 3D semantic labels [150] to filter out dynamic objects. We first extract the provided 3D bounding boxes of dynamic objects by thresholding the trajectory length of each labeled object. Afterwards, we extract the LiDAR points within the bounding boxes of dynamic objects from each LiDAR scan. For each training image, we project the extracted dynamic LiDAR points from its temporally nearest LiDAR scans onto the image space to form a binary mask, which indicates the pixels occupied by the dynamic objects. We also removed the dynamic LiDAR points from our LiDAR maps. Finally, we apply the mask to the training loss in Eq. 6.6. The results are shown in Fig. 6.6.

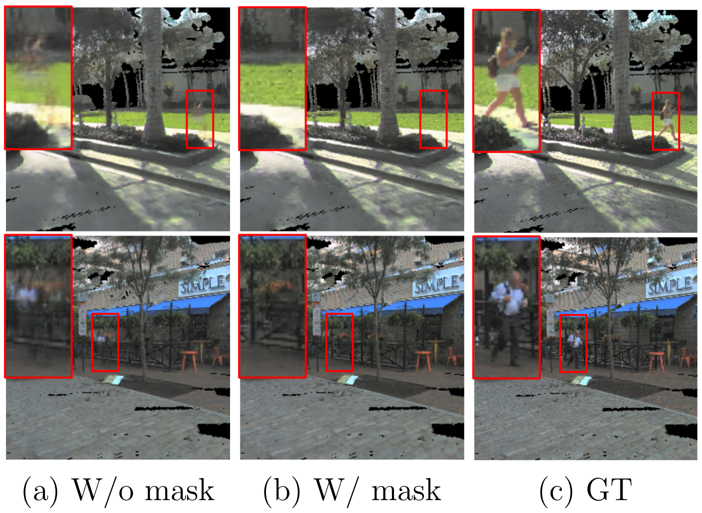


Figure 6.6: The results of w/o, w/ the moving object masks, and the ground truth. We observed the blurry shadows from moving objects in (a) and their removal in (b).

6.3 Experiments

6.3.1 Datasets

We extracted 8 sequences from the Argoverse 2 dataset [150]. Each sequence contains 3 cameras (front, front-left, front-right). The sequences were selected to avoid crowded scenes and scenes with large number of moving objects. The image contents are mostly city scenes with different ratio of artificial buildings and natural textures. We accumulated the LiDAR scans of the training images with provided ground truth poses to form the LiDAR maps. The LiDAR scans were collected at 10 Hz by two VLP-32C LiDARs with 64 beams in total. Since the original Argoverse 2 dataset only provides 10 Hz LiDAR poses and 20 Hz imagery without assigned poses, we first extracted the 10 Hz posed imagery by using the temporally closest LiDAR poses as image poses to reconstruct a visual map. And then we registered the other half of unposed images to the visual map with COLMAP to obtain totally 20 Hz image poses that match the LiDAR pose scale.

We subsampled one in every four images from the original 10Hz posed imagery for validation, and added the images with COLMAP poses to the training set. In total we have 155 training images and 22 validation images for each sequence. The

Table 6.1: Dataset statistics for Argoverse 2 sequences

Argoverse 2 dataset				
log id	**# train**	**#val**	**traj.** (m)	**# LiDAR points**
0a13	155	22	22.9	502,567
2aea	155	22	26.3	517,136
2b04	155	22	6.5	338,572
3e7c	155	22	31.0	547,837
4d7b	155	22	19.9	298,121
4d32	155	22	28.4	415,469
42c8	155	22	31.6	580,788
4690	155	22	28.4	546,343

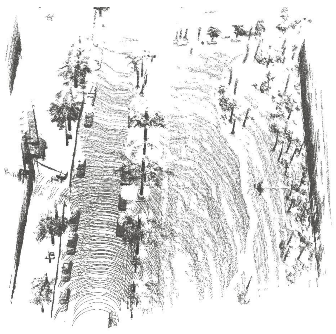
number of points in our LiDAR maps range from 3.0×10^5 to 5.8×10^5 and the camera trajectory lengths span from $6.49m$ to $31.6m$. Note that we collected images from 3 front-facing cameras in contrast to the 12 ring camera setting in [131], and our LiDAR point cloud is much sparser than [107]'s. These makes our dataset more challenging than previous works. Some statistics of the collect Argoverse 2 sequences, including the number of train/val images, trajectory length, and number of LiDAR points in the map, are shown in Tab. 6.1. Visualizations of the LiDAR maps are shown in Fig. 6.7.

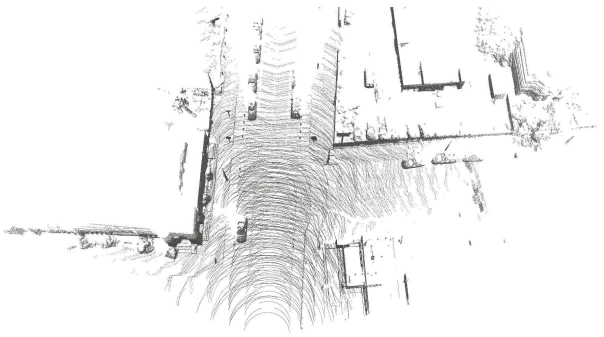
To form a more challenging case, we added LiDAR rain noise to individual scans according to [52] before building the map. We used $R = 8$ and $z_{max} = 200$ as the parameters. This noise model consists of two parts: 1) a threshold that removes faraway LiDAR measurements in the rain with minimum detectable power and a LiDAR intensity decaying model in the rain. 2) a rain noise model for LiDAR range measurement. The resulting maps are shown in Fig. 6.8.

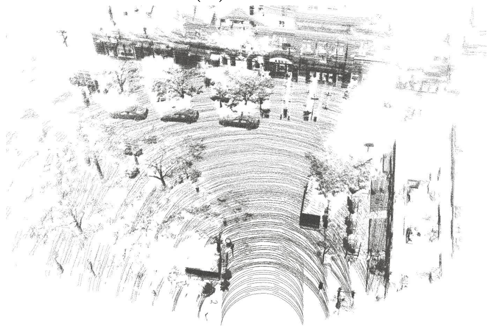
6.3.2 Baselines, Metrics, and Implementation

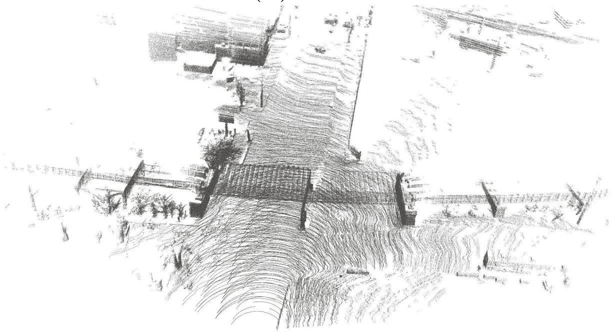
We compared with the point-based NeRF [154] and the outdoor state-of-the-art BlockNeRF [131].

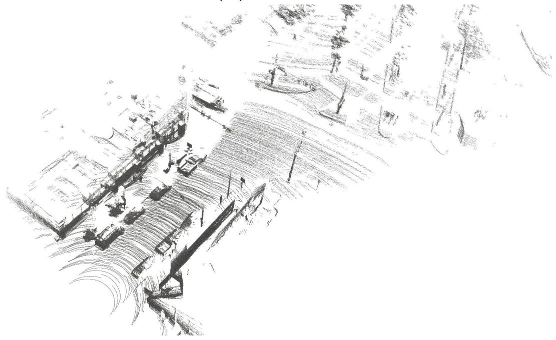
Point-based NeRF. We compared our final results with our volume rendering pipeline output, which is based on [154], but with view direction dependency and point cloud refinement module removed. We kept the rest of the pipeline (e.g. point

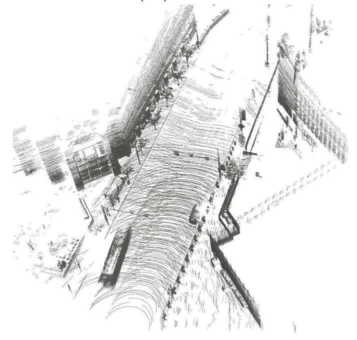

(a) 0a13

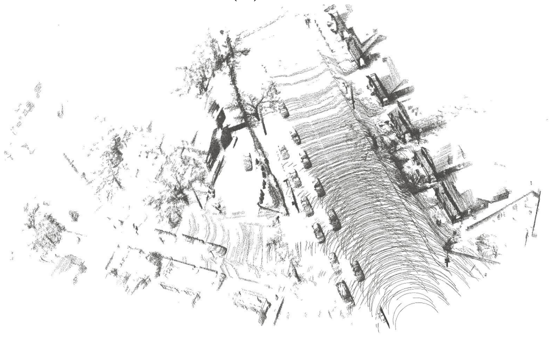

(b) 2aea


(c) 2b04


(d) 3e7c


(e) 4d7b


(f) 4d32


(g) 42c8

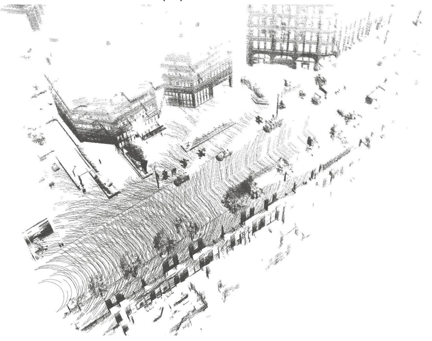

(h) 4690

Figure 6.7: Visualization of the collected LiDAR maps from Argoverse 2 dataset.

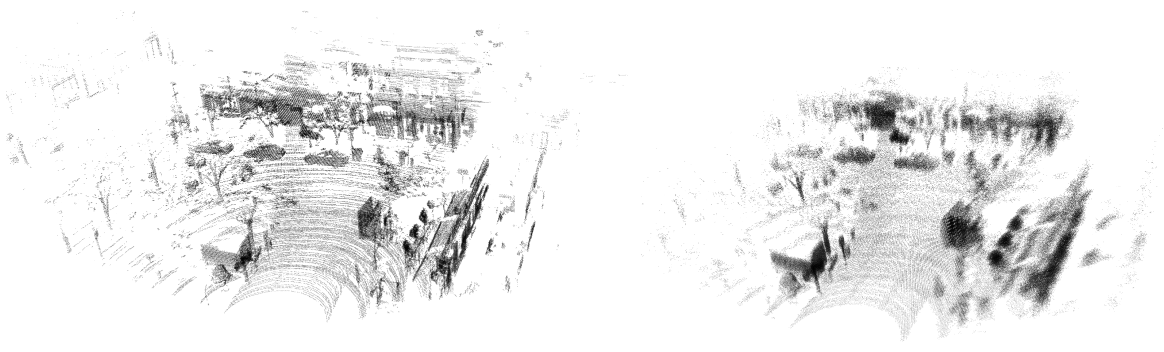


Figure 6.8: The LiDAR maps before (a) and after (b) adding rain noise. The LiDAR measurements affected by the rain noise has shorter and noisier range.

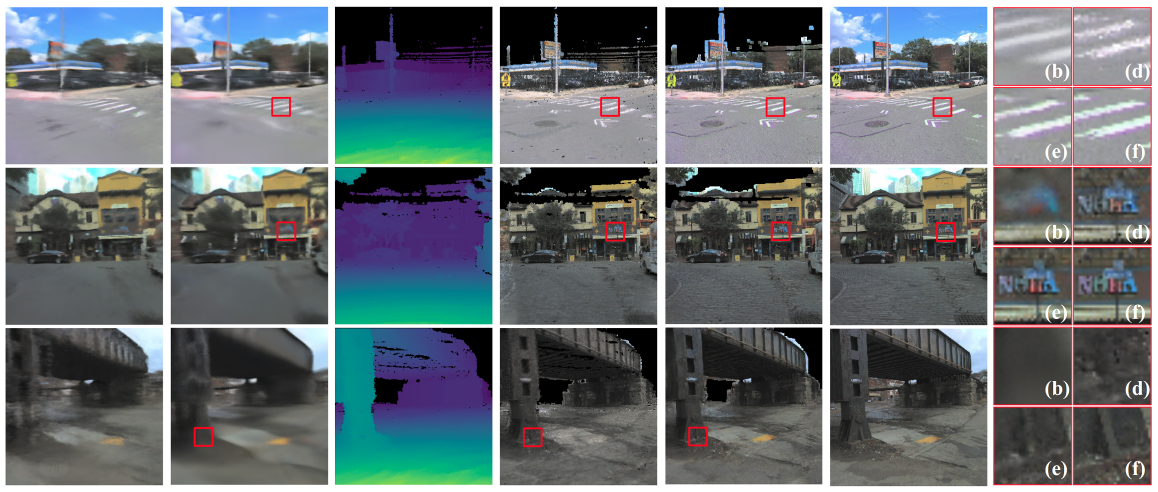


Figure 6.9: Visual comparison with baselines. (a) BlockNeRF (b) BlockNeRF + depth (c) our depth (d) point-based (e) our RGB (f) GT (g) enlarged patches. From the enlarged image patches (g) we can observe that our results (e) have better image quality and are visually closer to the ground truth patches (f).

sampling, weighting functions) the same as our system except for the refinement module for fair comparison. We applied the same LiDAR depth and $\mathcal{L}_1$ losses to the point-based NeRF output. Each model was trained with Adam optimizer using one whole image as one batch and learning rate 1×10^{-4} for 10000 epochs until convergence.

BlockNeRF [131] + depth [107]. For the image-only baseline, we compared with [131], the state-of-the-art outdoor large-scale NeRF. The longest camera trajectory of our collected Argoverse 2 sequences ($31.6m$) is within a reasonable range of block size in [131], so we used one block for each sequence. Furthermore, we also compared with the BlockNeRF with LiDAR depth supervision [107] for fair comparison. Because the original code of both [131] and [107] are not available, we ran experiments with an unofficial BlockNeRF implementation, and incorporated the depth loss as described in URF [107]. Each model was trained with Adam optimizer with learning rate 5×10^{-4} and batch size 1024 for 500 epochs, where the validation error converged.

Metrics. We compared PSNR, SSIM, LPIPS as in [93]. We masked out the regions without LiDAR depth (the black regions in the volume rendering outout) since our pipeline does not focus on generating those pixels (Sec. 6.5). The same masks were applied to all the compared results when computing the metrics for fair comparison. Otherwise the evaluation result would be diluted by the large black regions. Note that our numbers are not directly comparable to the numbers reported by previous works due to these masks.

Implementation details. We first built a k-d tree from our LiDAR maps, and queried the nearby LiDAR points to ray samples with the k-d tree. After obtaining the set of ray samples and kNN LiDAR points, we implemented our volume rendering with DGL [142] and PyTorch. A heterogeneous local graph was built with pixels, ray samples and LiDAR points as nodes. Different types of edges connect pixels to the corresponding ray samples, and ray samples to the nearby LiDAR points. The MLPs and the aggregation function were implemented as graph functions. We used $K = 10$, $M_0 = 1000$, $M = 16$, $\zeta = 0.15m$, $\beta = 10$, $\mu = 0.05m$, $\lambda = 100$, $F_m = 8$, and $F_h = 32$ as design parameters. The numbers of MLP layers in $\mathcal{F}$, $\mathcal{F}_\sigma$, and $\mathcal{F}_\mathbf{c}$ are all 3. The parameters and network structures here are tuned with clean Argoverse 2 sequences and applied to all the datasets. Our current system takes about $5s$ to render an image. For each sequence, we trained independent models with the whole pipeline in an end-to-end fashion for 1000 epochs with Adam optimizer, using one whole image as a batch and learning rate 1×10^{-4}. It takes about 0.5 days to train a model on a single GeForce RTX 3090 GPU at image resolution of 256×256.

6.3.3 Comparison with Baselines

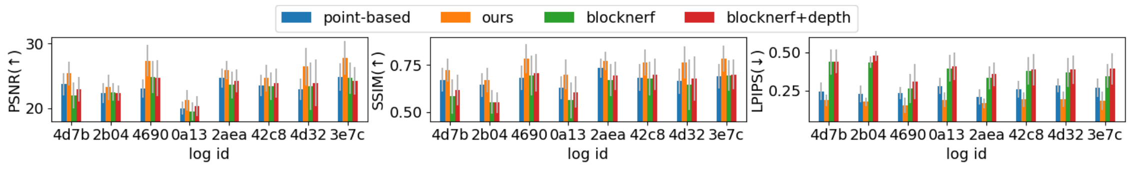
We observed that the proposed 2D refinement strategy outperformed point-based NeRF by a large margin in every sequence tested (Fig. 6.10). From the rendered images in Fig. 6.9 (d)(e), we observed significant image quality improvement with the proposed image refinement stage. The refinement module not only reduced the noise but also rendered details better. Besides, we obtained very blurry results from BlockNeRF even with LiDAR depth supervision, as shown by the metrics in Fig. 6.10 and the visualizations in Fig. 6.9 (a)(b). Incorporating the LiDAR depth supervision significantly improved the depth map quality from BlockNeRF (Fig. 6.4 (c2)) but the rgb result is still blurry. Although the rendered image area of BlockNeRF is not as limited as ours (e.g. the sky), it would require specially collected datasets with better view coverage to improve this result. This drawback of image-only NeRF was also pointed out by [34, 107, 109].

6.3.4 Resistance to Noise

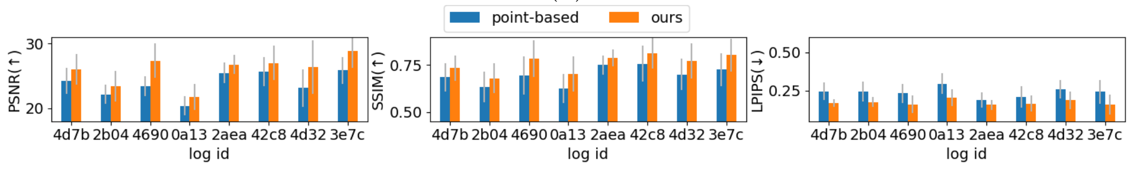
Although the LiDAR measurements are relatively more accurate than SfM, they can degrade greatly in bad weather conditions. We simulated such LiDAR noise in rainy days [52] before accumulating single LiDAR scans into LiDAR maps, to evaluate the resistance to noise (Fig. 6.8). Quantitative evaluation for these harder cases are shown in Fig. 6.10. We observed that our method still outperformed the purely point-based baseline in the evaluated metrics and also recovered reasonable visual details as shown in Fig. 6.11.

6.3.5 Ablation Study for cGAN Loss

The proposed cGAN refinement module not only significantly improves the rendered image quality, but also provides the flexibility for users to select desired level of image



(a)



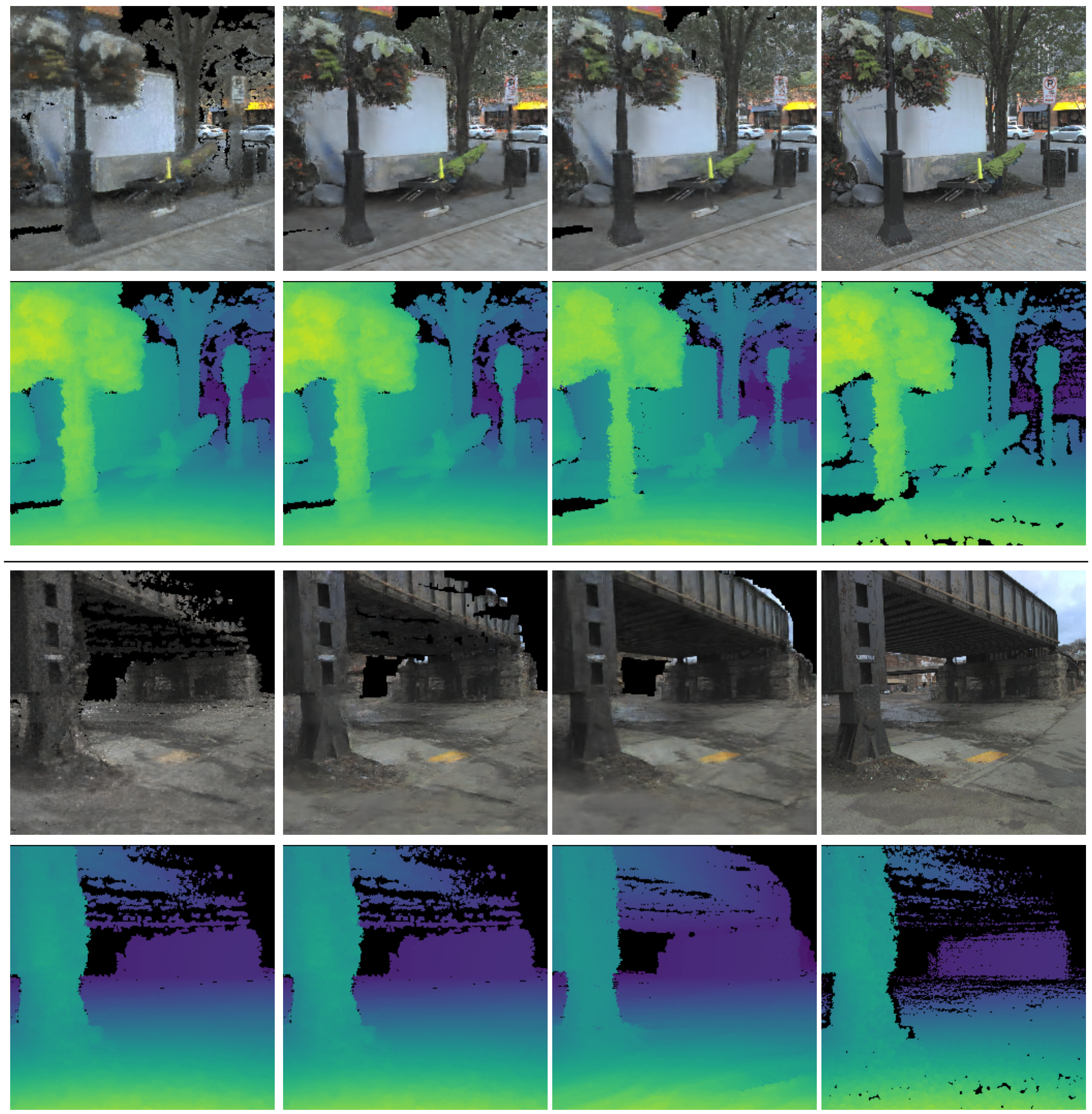
(b)

Figure 6.10: Quantitative comparisons. (a) Our method significantly outperformed the baselines. The BlockNeRF results are very blurry, as reflected by the high LPIPS scores. (b) Results with noisy LiDAR maps.

details. In this experiment, we applied different weights ω_{cGAN} to $\mathcal{L}_{\text{cGAN}}$ in Eq. 6.6 to observe the effect of cGAN loss to image quality. The quantitative results and visualizations are shown in Fig. 6.12 and 6.13. The results reveal interesting trade-off between PSNR and LPIPS: when increasing ω_{cGAN}, the PSNR decreases but LPIPS improves (decreases). This leads to several open research questions: what is the best image metric to use and how to find the optimal balance among multiple image metrics? How to design the training losses to obtain the most desirable balance of image metrics? The answer could be application-oriented and is worth more future research.

6.4 Applications

Although many NeRF works have been published since 2020, most of the advancements were for improving image quality. Recently, more and more researchers start to explore wider uses such as robot navigation and SLAM [3, 110]. This trend implies the great yet not well-explored potential of using NeRF in more applications. From this application-oriented standpoint, we should evaluate the NeRF outputs not only based on image quality, but also the performance in targeted applications, as it is what matters in the end.



(a) Point-based (noisy) (b) Ours (noisy) (c) Ours (clean) (d) GT (noisy)

Figure 6.11: Visual comparison with baselines on noisy LiDAR maps.

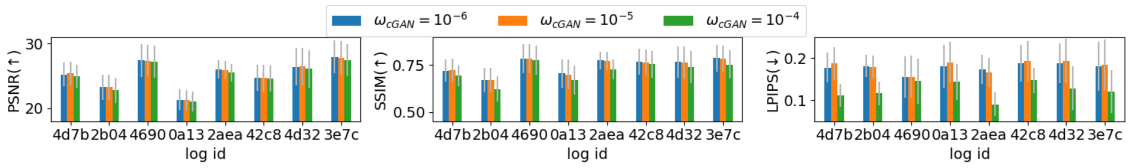


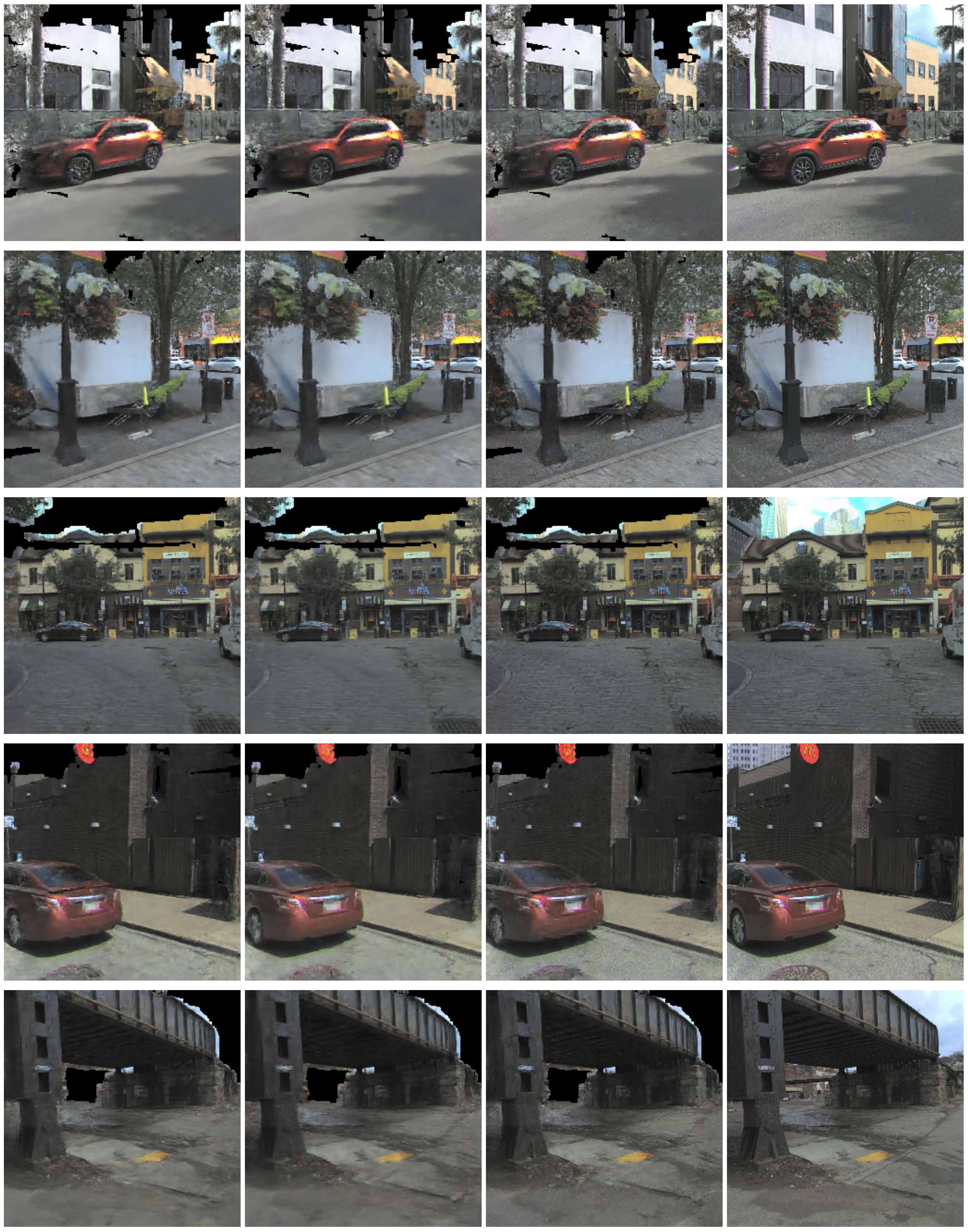
Figure 6.12: Ablation study for cGAN loss strength. We observed a trade-off between PSNR and LPIPS. Stronger cGAN loss adds more details to the output image, potentially making the image more perceptually pleasant with respect to LPIPS, but not necessary faithful with respect to PSNR.

6.4.1 Object Detection Simulator

We compared the detection results on our synthetic images and the ground truth images using Detectron2 [153]. Considering that the car class usually dominates autonomous driving applications, we performed this comparison on two logs with more visible vehicles (log 0a13 and 4d7b). Our results show that the detected object masks are visually similar on synthetic and ground truth images, and our mean IoU for the car class among the validation images outperformed the baselines (Fig. 6.14 and 6.15 (a)). This result indicates that the synthetic images rendered by our system can potentially be used to further simulate the detection-related robot behaviors, such as object tracking when robot is moving, or path-planning considering the existence of other objects.

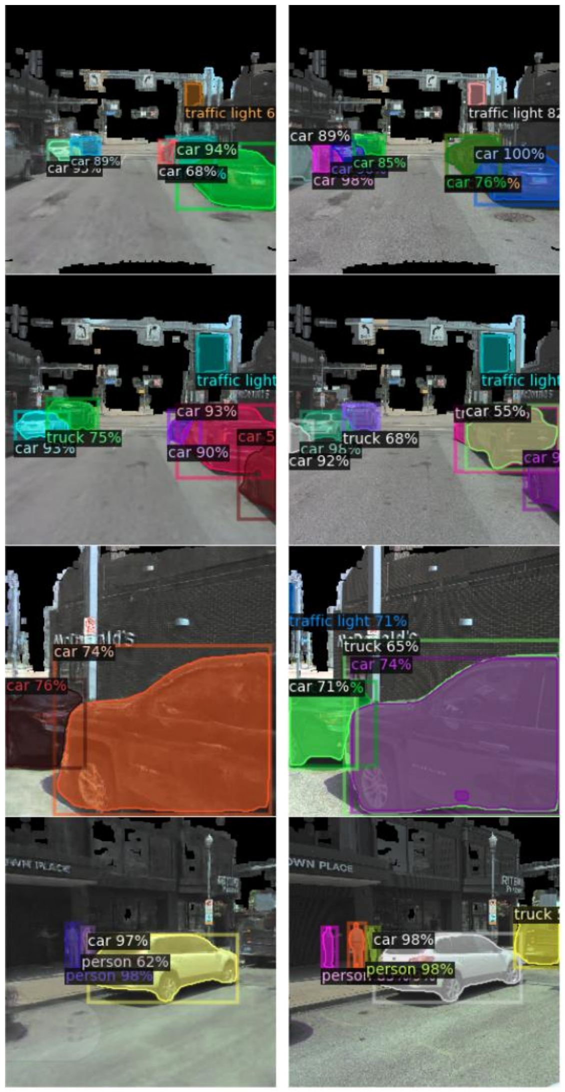
6.4.2 Data Augmentation

Data collection is essential for many deep neural network applications. However, collecting real-world data can be an expensive and time-consuming process, because it typically requires driving a mobile robot across the desired environment for multiple passes. One possible way to lighten the burden is to first train a NeRF with some collected images, and augment the dataset with synthetic images rendered by NeRF. This is especially useful for data-hungry deep networks such as pose regression networks, whose performance largely depend on the training set size [119]. Inspired by [94], we augmented our collected Argoverse 2 sequences (Sec. 6.3.1), and compared the MapNet [12] performance with the same real validation set before and after the augmentation. The MapNet network takes a RGB image as input and predicts a

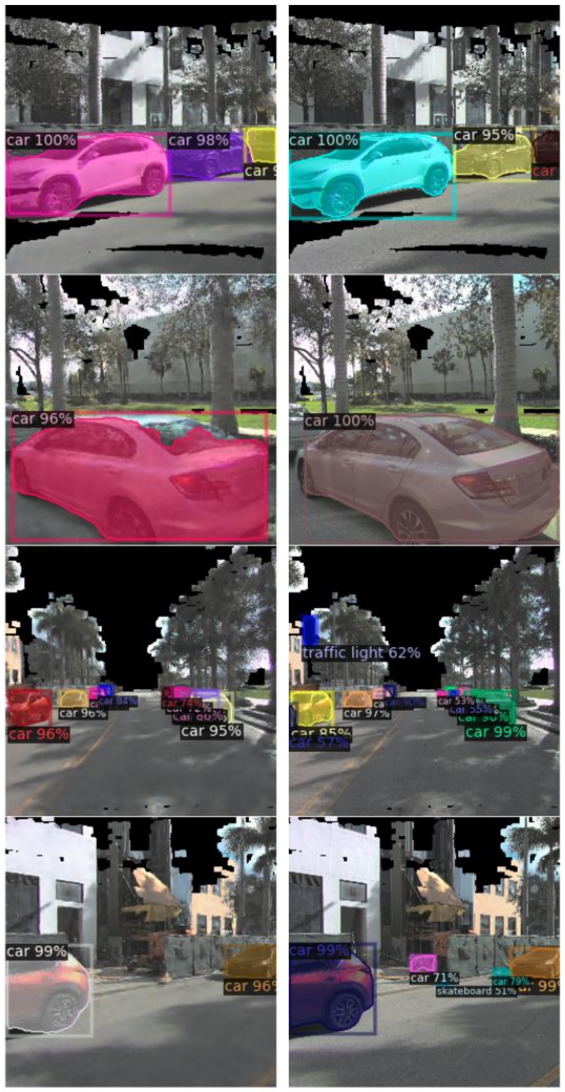


(a) $\omega_{\text{cGAN}} = 10^{-6}$ (b) $\omega_{\text{cGAN}} = 10^{-5}$ (c) $\omega_{\text{cGAN}} = 10^{-4}$ (d) GT

Figure 6.13: The effect of cGAN loss strength. The results applying different cGAN loss strength are shown in (a)-(c). We can observe that the images with larger ω_{cGAN} contain more details but not necessary faithful to the ground truth.

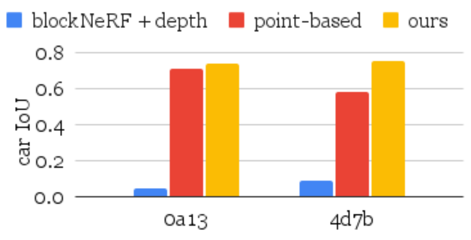


(a) Left: ours/ right: GT (log 4d7b)

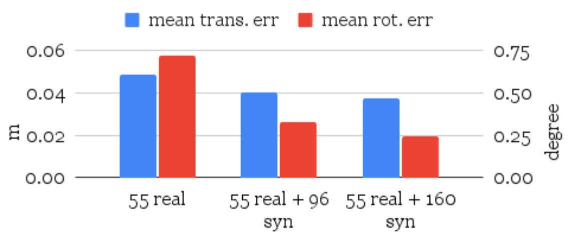


(b) Left: Ours/ right: GT (log 0a13)

Figure 6.14: Side-by-side comparisons of the Detectron2 results on our synthetic images and ground truth images from the validation set.



(a) Car IoU



(b) Pose prediction errors

Figure 6.15: (a) Our rendered images get higher car IoU against GT than the baselines. (b) Data augmentation with our synthetic images significantly reduced the pose prediction errors of MapNet.

6-DoF camera pose. For augmentation, we added uniformly random noises within range $[-0.5, +0.5]m$ in xz for translation, and $[-3, +3]$ degree in yaw for rotation. The results in Fig. 6.15 (b) show significantly lower pose errors in the cases with data augmentation, indicating that our synthetic images provide useful information for MapNet despite the black regions.

We compared three cases in this experiment: 1) 52 real images per sequence, 2) 52 real images + 96 synthetic images, 3) 52 real images + 160 synthetic images. For each case we trained MapNet [12] until the validation loss stopped improving for 8000 epochs. The detailed quantitative results are shown in Tab. 6.2 and the corresponding mean validation loss curve is shown in Fig. 6.16. The validation loss in Fig. 6.16 is a combination of translation and rotation losses, as described in [12]. The results show that training with more synthetic images improves the results with less pose errors and validation loss. One exception we observed is that the log 2b04 got the best result in case 2 instead of case 3 unlike other sequences, which is caused by the training process of case 3 stuck in a local minimum. Overall, the results show significant benefit of this realistic and geometry-based data augmentation for pose regression.

6.4.3 Changing Seasons

If trained with seasonal information, the proposed system can perform season changing visual effect. With this ability, we can potentially use our system as a simulator with the control of seasonal appearance. We performed this experiment with the NCLT

Table 6.2: MapNet results with different levels of data augmentation, where the median and mean for translation and rotation pose prediction errors are represented as $t_{med}(m)$, $t_{mean}(m)$, and $r_{med}(°)$, $r_{mean}(°)$. The cases with augmented images significantly outperformed the case trained with only real images.

	52 real				52 real + 96 syn.				52 real + 160 syn.			
log id	t_{med}	t_{mean}	r_{med}	r_{mean}	t_{med}	t_{mean}	r_{med}	r_{mean}	t_{med}	t_{mean}	r_{med}	r_{mean}
0a13	0.03	0.05	1.01	1.2	**0.02**	0.03	**0.17**	0.37	0.03	0.04	0.2	**0.2**
2aea	0.02	0.04	0.42	0.63	0.02	0.03	0.34	0.35	0.02	0.03	**0.15**	**0.2**
2b04	0.04	0.05	0.24	0.28	**0.01**	**0.03**	**0.21**	**0.21**	0.02	0.04	0.44	0.42
3e7c	0.04	0.07	1.82	2.2	0.04	0.04	**0.12**	0.24	**0.03**	**0.03**	0.18	**0.21**
4d7b	0.02	0.03	0.23	0.24	0.02	0.03	0.29	0.44	0.02	**0.02**	**0.16**	**0.21**
4d32	0.04	0.07	0.45	0.44	0.05	0.07	0.29	0.4	0.04	**0.06**	**0.2**	**0.22**
42c8	0.03	0.04	0.35	0.44	0.03	0.05	0.31	0.33	0.03	0.04	**0.24**	**0.29**
4690	**0.03**	0.04	0.36	0.34	0.04	0.04	0.3	0.29	0.04	0.04	**0.15**	**0.18**

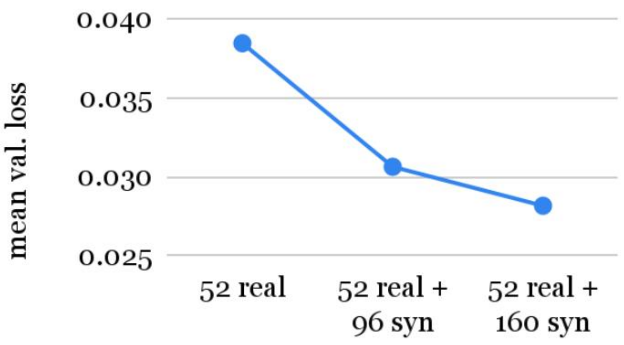


Figure 6.16: Mean validation loss for different data augmentation setups. Increasing the amount of augmented data significantly reduces the validation loss.

Table 6.3: Dataset statistics for NCLT sequences

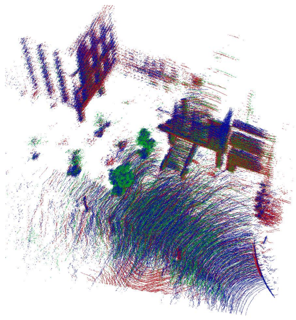
NCLT dataset				
log id	**# train**	**#val**	**traj.** (m)	**# LiDAR points**
area 1	159	75	22.0	841,587
area 2	137	67	17.8	721,461
area 3	167	78	23.1	1,352,948

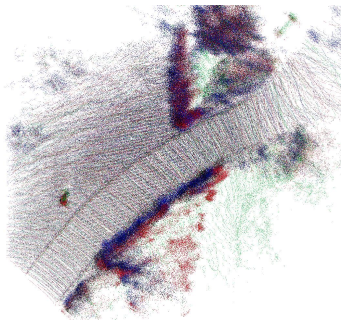
dataset [17], a long-term camera-LiDAR dataset that was collected through different seasons. The multi-season camera-LiDAR sequences were extracted from NCLT, each with multiple passes of similar trajectories in different seasons. We used the 5Hz imagery from one front-facing camera, and selected one in every three frames for validation. Three sequences were extracted in total, with each containing 137-167 training images and 67-78 validation images.

The LiDAR scans were collected with a Velodyne HDL-32E LiDAR. The number of points in the LiDAR maps range from 7.2×10^5 to 1.4×10^6, and the camera trajectory lengths span from $17.8m$ to $23.1m$ (Tab. 6.3). For each training image, we extracted the LiDAR scan with closets timestamp and used the extracted LiDAR scans to build the LiDAR map (Fig. 6.17). Visualizations of the results from the three collected sequences are shown in Fig. 6.18. One can observe that the noise level is lower in Argoverse 2 maps (Fig. 6.7) than NCLT dataset (Fig. 6.17). This was caused by more accurate LiDAR pose estimation and has motivated us for using the tight sampling strategy in the Argoverse 2 dataset.

For the appearance embedding, we used a scalar $t \in [0, 1]$ to represent the normalized timestamp when the data was recorded (e.g. $t = 0.33, 0.66, 1$ means spring, summer and winter). We first encoded t with positional encoding, and used the encoded t, $\mathbf{t}_j$ with an AdaIN [62] module to incorporate $\mathbf{t}_j$ into $\mathbf{h}_j$ (the blue box in Fig. 6.2 (b)). After training, we applied different t values to a given validation view pose to obtain different seasonal appearances (Fig. 6.18).

6.5 Limitations


(a) Area 1


(a) Area 2

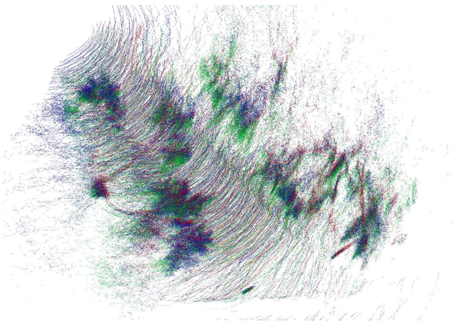

(c) Area 3

Figure 6.17: Visualization of the collected LiDAR maps from NCLT dataset. LiDAR points collected in different seasons are shown by different colors. The LiDAR maps from NCLT consist of LiDAR points collected from different seasons. One can also observe the seasonal foliage shape change. For example, the green points in (c) were collected in August and spread wider than the points collected in other seasons, reflecting the fact that the shape of seasonal foliage is larger in summer.

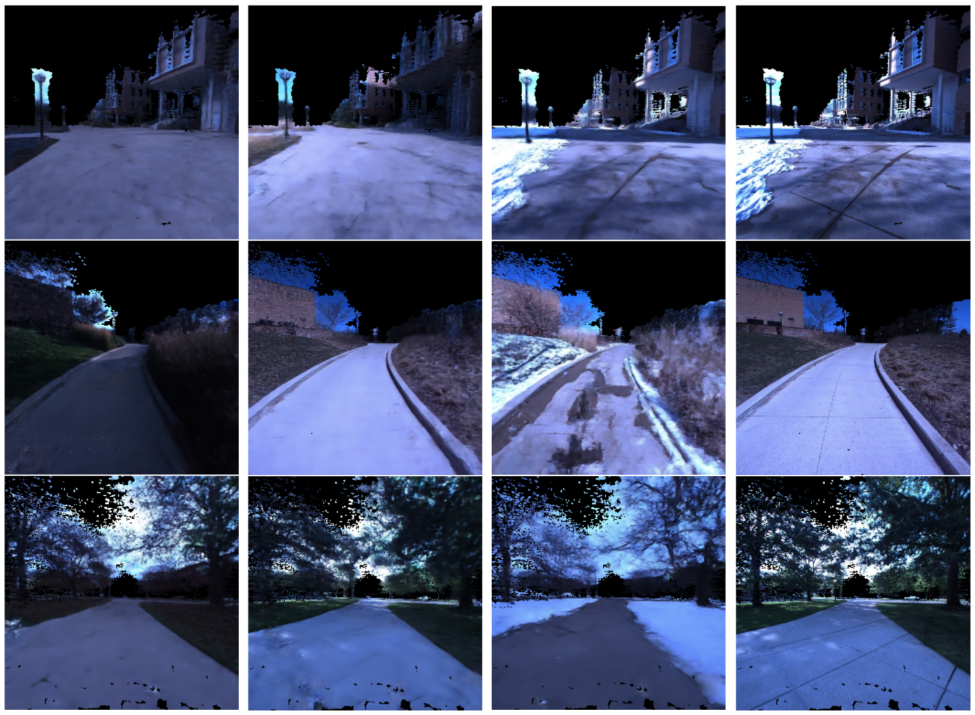

(a) Season 1 (b) Season 2 (c) Season 3 (d) GT

Figure 6.18: Visualization of changing season results for the corresponding LiDAR maps in Fig. 6.17. From top to bottom are the results from area 1, 2, and 3.

Our pipeline skips and returns black color at the pixels where the LiDAR depth is unavailable. The same limitation was presented in PointNeRF [154]. The black regions are mostly on the sky, top of tall buildings, and objects too far away. Another current limitation is that non-lambertian surfaces, such as transparent and reflective objects, are currently unmodeled. This can potentially be improved by introducing latent variables to model these materials in the volume rendering MLPs.

For now our implementation relies on accurate camera-LiDAR calibration, therefore optimizing the calibration parameters with NeRF can also be a future work. Also, here we focus on offline image rendering quality but runtime still has room to improve. For example, we expect the point embeddings to be repetitive and can be pruned and compressed. Besides, large moving objects with inaccurate 3D labels might cause bad results.

Finally, we visualize some failure cases for future research reference (Fig. 6.19). Our system can produce unsatisfactory results when given inaccurate 3D labels and LiDAR depth. Also, thin objects are still challenging to render.

6.6 More Results

In this section, we show more visualizations from our method and the baselines. The depth and RGB outputs from BlockNeRF with URF LiDAR depth loss, the point-based baseline, and our method are show in Fig. 6.20, 6.21, 6.22, and 6.23. Resonating the findings of previous NeRF works, our study also showed that the positional encoding module is helpful (Tab. 6.4). The contribution of the proposed point-sampling strategy is shown quantitatively and qualitatively in Tab. 6.4 and Fig. 6.24.

6.7 Conclusion

The proposed pipeline performs high-quality NVS from outdoor camera-LiDAR datasets. Our system combines the strength of LiDAR sensors, NeRF, and cGAN, and outperforms the baselines significantly. We believe that the practical use of NeRF is an uprising research area, and look forward to future developments in this field.

Table 6.4: Ablation study. We show the contribution of each component quantitatively. The use of cGAN significantly improved results with all image metrics. The proposed tight point sampling strategy and the positional encoding module also helped. Note that the noisy and clean versions have different black regions and the numbers are not directly comparable.

map type	cGAN	tight sampling	$\gamma(.)$	ω_{cGAN}	PSNR (↑)	SSIM (↑)	LPIPS (↓)
clean		✓		$1e-5$	22.60	0.65	0.28
	✓	✓		$1e-5$	25.02	0.74	0.19
			✓	$1e-5$	23.02	0.67	0.25
	✓		✓	$1e-5$	25.19	0.74	0.18
		✓	✓	$1e-5$	23.14	0.67	0.25
	✓	✓	✓	$1e-5$	25.24	**0.75**	0.18
	✓	✓	✓	$1e-4$	24.94	0.72	**0.13**
	✓	✓	✓	$1e-6$	**25.28**	**0.75**	0.18
noisy		✓	✓	$1e-5$	23.79	0.70	0.24
	✓	✓	✓	$1e-5$	**25.93**	**0.76**	**0.17**

Table 6.5: Quantitative number comparison for the Argoverse 2 sequences

map type	method	metric	log ID							
			4d7b	2b04	4690	0a13	2aea	42c8	4d32	3e7c
clean	point-based [154]	PSNR	23.73	22.39	23.07	20.01	24.67	23.49	22.97	24.87
		SSIM	0.67	0.64	0.68	0.63	0.73	0.68	0.66	0.69
		LPIPS	0.25	0.23	0.24	0.28	0.21	0.26	0.28	0.27
	BlockNeRF [131]	PSNR	22.03	22.50	24.78	19.50	23.59	23.41	23.45	24.66
		SSIM	0.58	0.55	0.70	0.56	0.67	0.68	0.64	0.70
		LPIPS	0.44	0.44	0.26	0.39	0.34	0.38	0.37	0.35
	BlockNeRF+depth [107, 131]	PSNR	22.90	22.32	24.67	20.32	24.23	23.88	23.93	24.23
		SSIM	0.62	0.55	0.71	0.61	0.69	0.70	0.68	0.70
		LPIPS	0.44	0.48	0.31	0.41	0.36	0.39	0.39	0.39
	ours	PSNR	25.38	23.23	27.34	21.24	25.88	24.68	26.43	27.78
		SSIM	0.72	0.67	0.78	0.70	0.77	0.76	0.76	0.79
		LPIPS	0.19	0.18	0.16	0.19	0.17	0.19	0.19	0.18
noisy	point-based	PSNR	24.24	22.15	23.44	20.37	25.47	25.66	23.15	25.87
		SSIM	0.68	0.63	0.69	0.63	0.75	0.76	0.70	0.73
		LPIPS	0.24	0.25	0.23	0.30	0.19	0.21	0.26	0.24
	ours	PSNR	25.98	23.40	27.34	21.77	26.75	27.01	26.37	28.79
		SSIM	0.73	0.68	0.79	0.70	0.79	0.82	0.77	0.80
		LPIPS	0.17	0.18	0.16	0.20	0.16	0.17	0.19	0.16

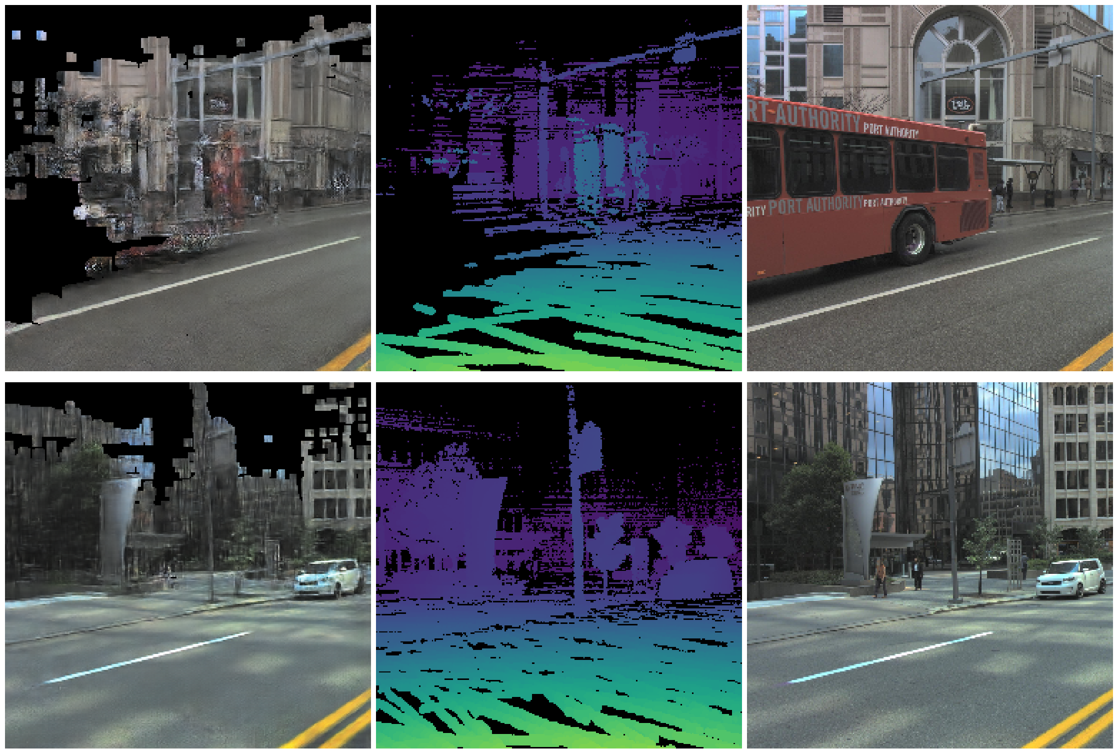


Figure 6.19: Some failure cases. (Top) Incomplete dynamic object removal caused by inaccurate 3D label. It leaves ghosts in the LiDAR map and affect our output. (Bottom) Thin objects with color similar to the background.

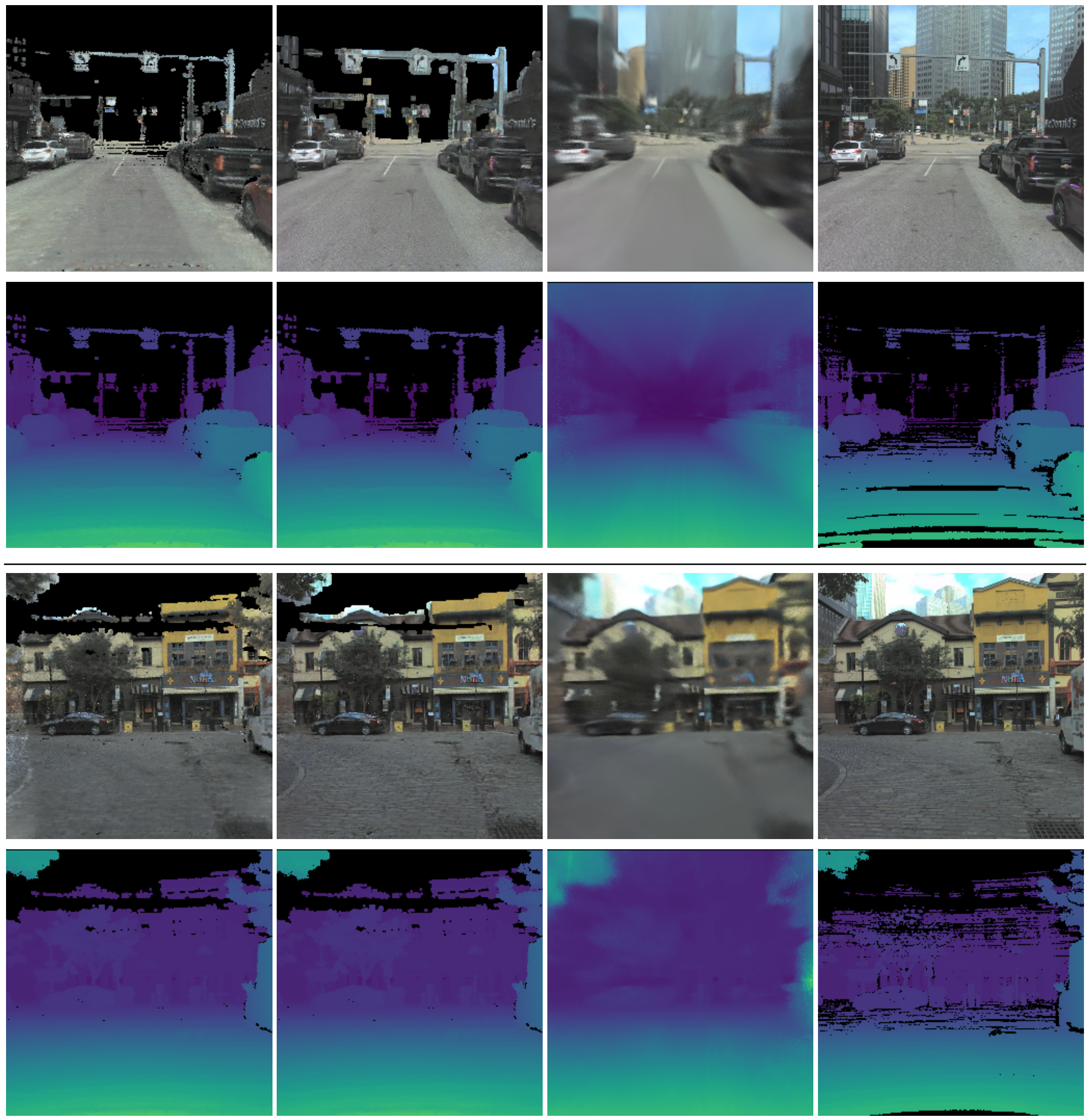

(a) Point-based (b) Ours (c) BN + depth (d) GT

Figure 6.20: Additional visual comparison with baselines

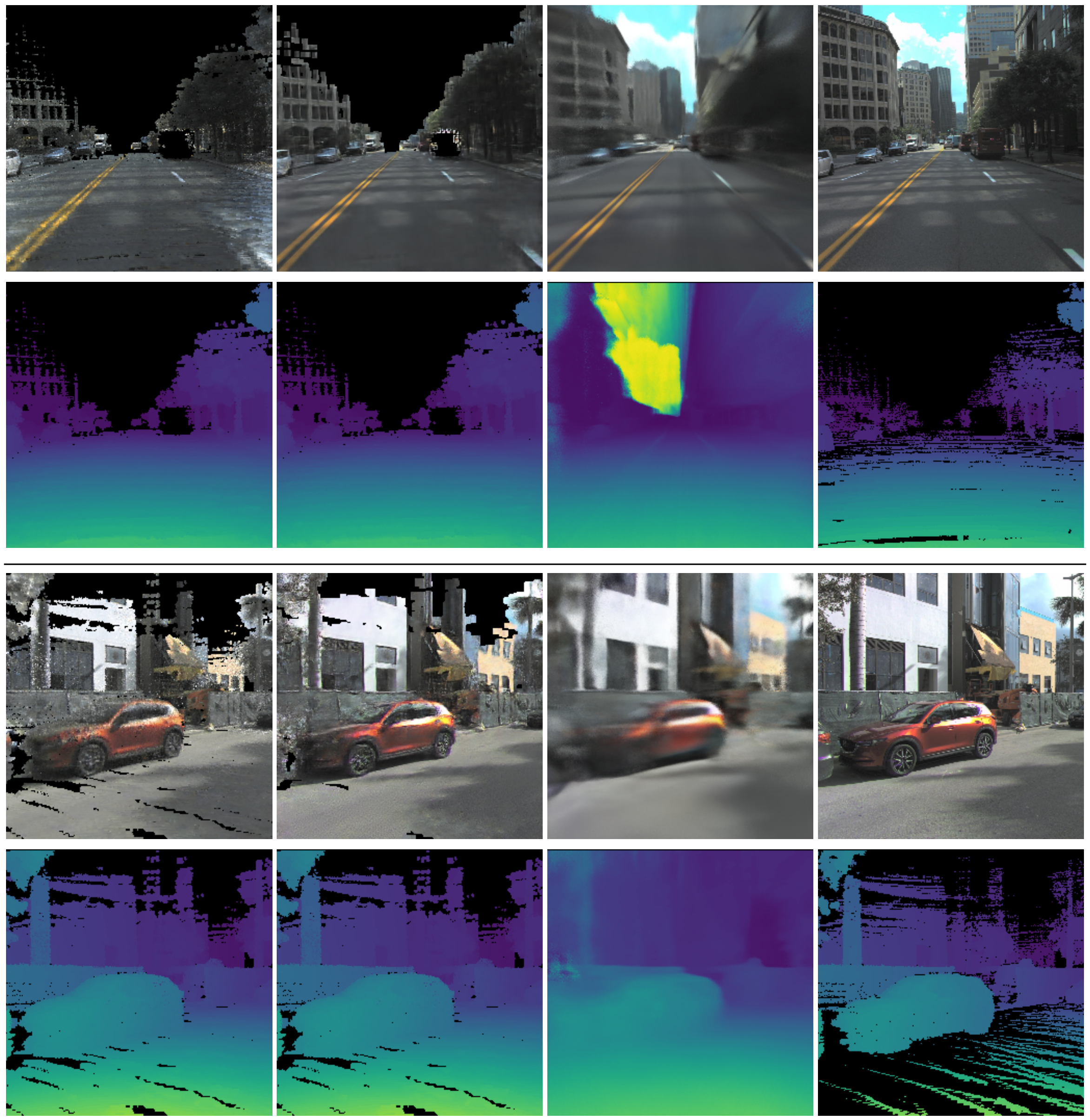
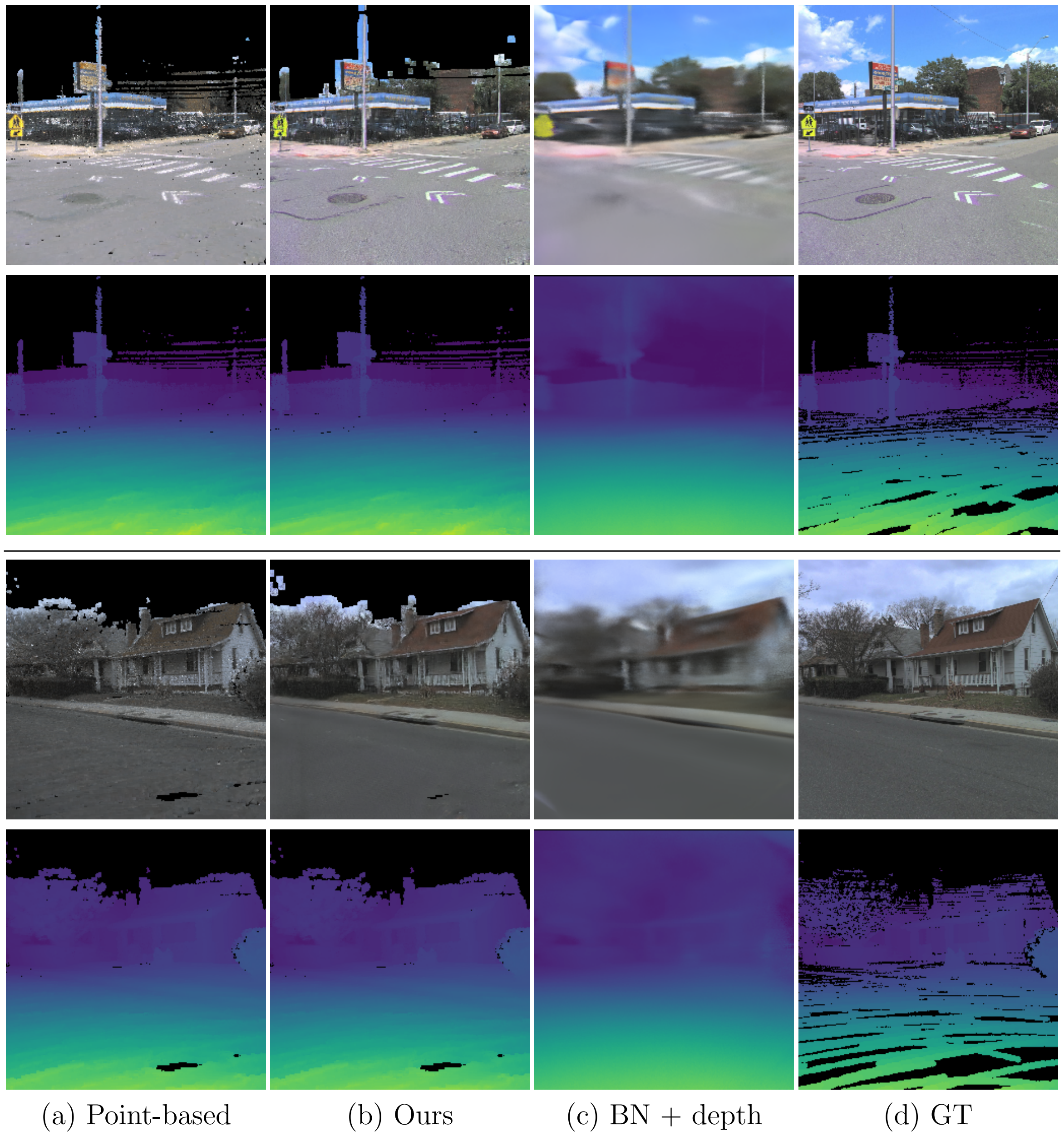


Figure 6.21: Additional visual comparison with baselines



(a) Point-based (b) Ours (c) BN + depth (d) GT

Figure 6.22: Additional visual comparison with baselines

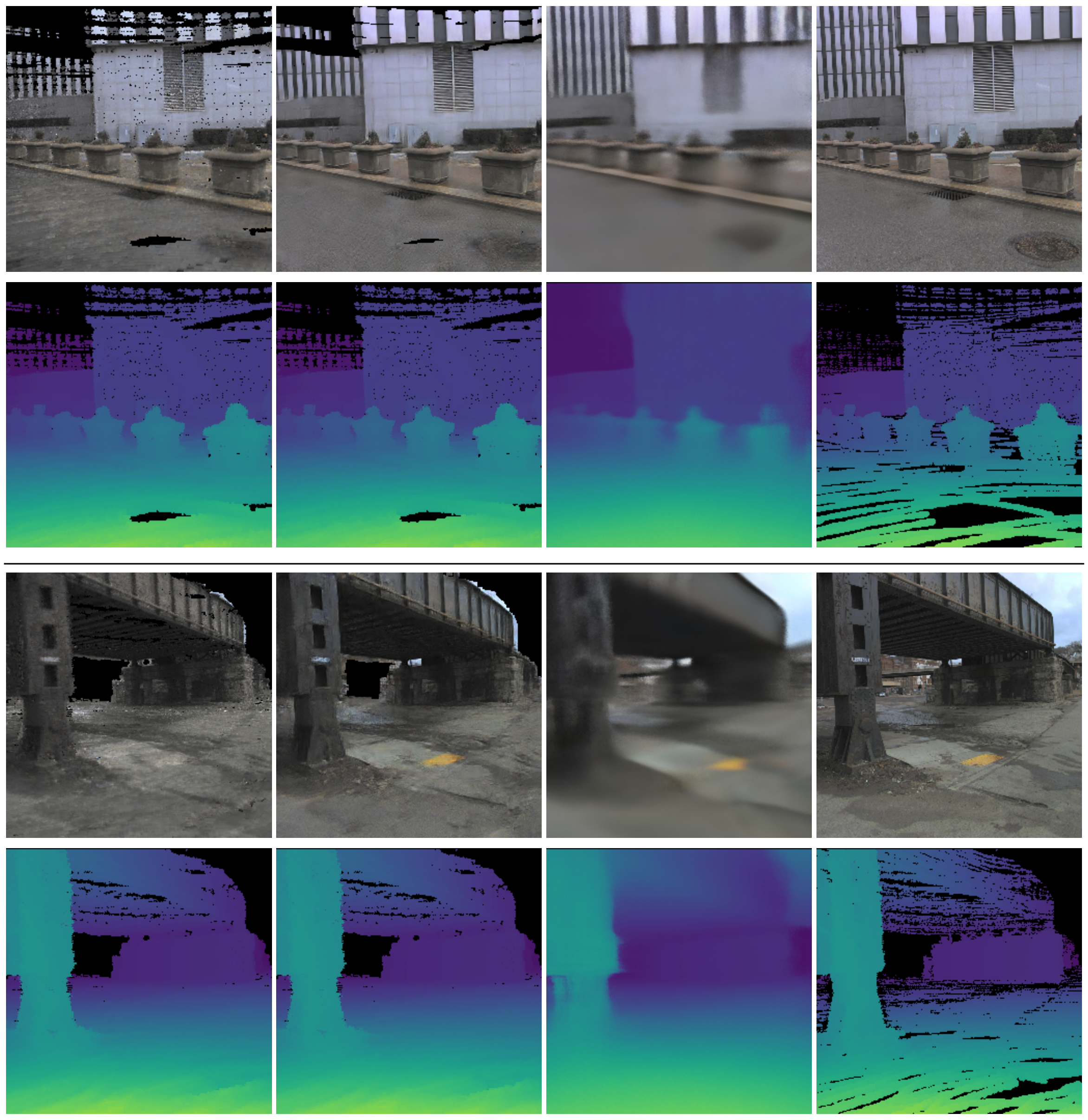
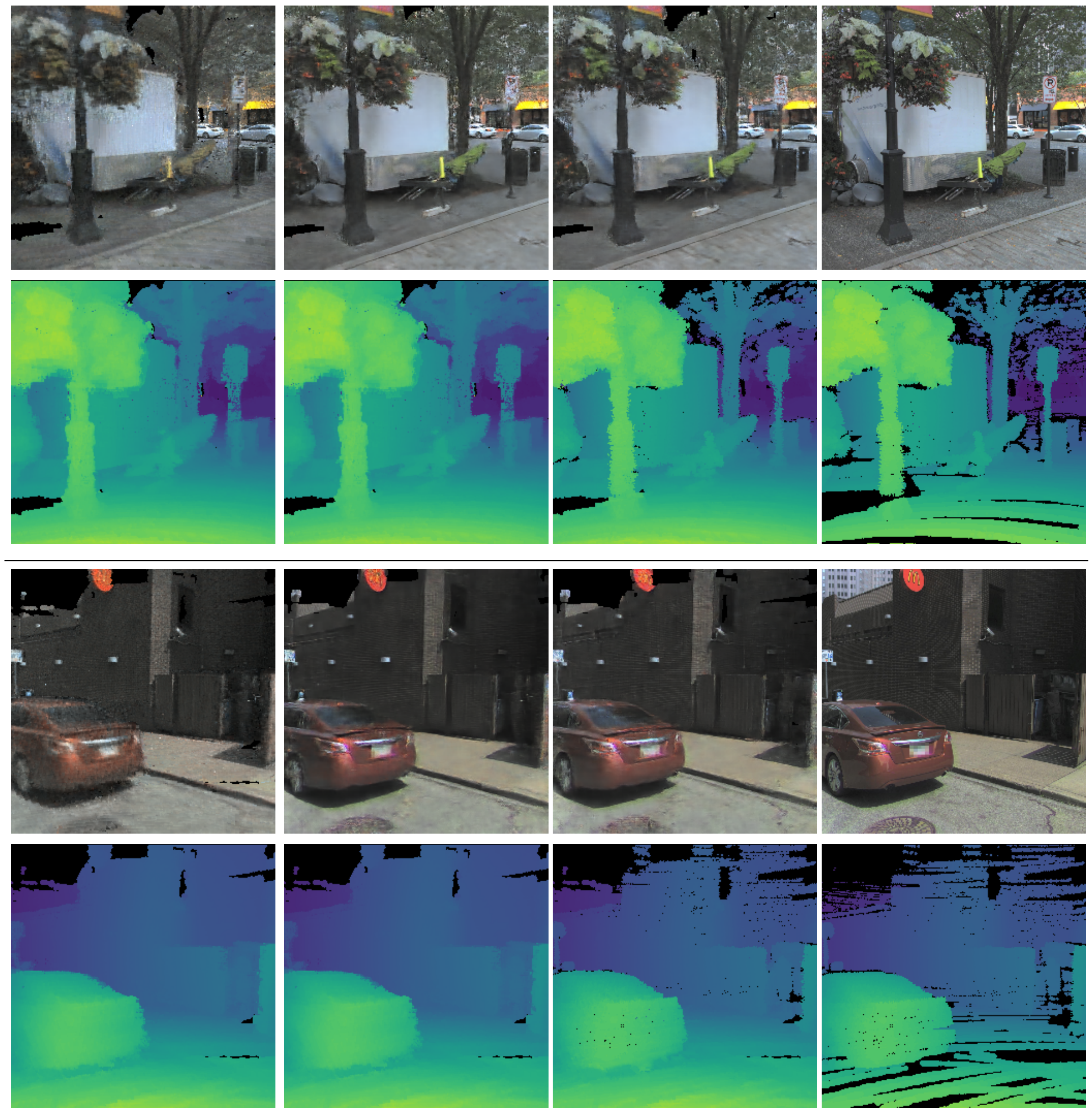

(a) Point-based (b) Ours (c) BN + depth (d) GT

Figure 6.23: Additional visual comparison with baselines



(a) Point-based (naive) (b) Naive (c) Ours (d) GT

Figure 6.24: Visual comparison for point sampling strategy. The proposed point sampling strategy gives more accurate depth than the naive radius based baseline.

Chapter 7

Conclusion and Future Works

In this thesis, we explored the map design problem from several perspectives: compression, sensor-fusion, and the combination of neural representation. We have described our works about map compression for image-to-LiDAR map, LiDAR-to-LiDAR map, and image-to-SfM map registration, and a LiDAR-assisted NeRF system for outdoor novel view synthesis. Our works combine deep learning techniques (end-to-end optimization on voxel grids, graph neural networks, NeRF) and classical knowledge (image and LiDAR registration, sensor fusion), and outperformed the corresponding baselines significantly. With deep appreciation to all of our collaborators, we shared our experiences in this thesis document and the corresponding publications [25, 26, 27, 28] with the hope of further bridging modern deep learning research and practical robot applications.

A common find in our works is the advantage of using multiple sensor modalities (e.g. camera, LiDAR). For example, the LiDAR maps used in Chap. 6 largely compensate the weakness of camera-only systems in low-textured, sparsely-sampled regions. The camera-to-LiDAR registration design described in Chap 3 greatly reduces the expected system hardware cost. In addition, the combination of learning-based and classical techniques generally exploits the strengths from both. Learned descriptors are shown to perform much better than hand-crafted descriptors (Chap. 4, 5), while the non-learning-based registration methods, such as ICP and PnP, are more robust and generalize better than end-to-end learned registration methods, and are used widely in real-world applications.

There are remaining problems to solve in the fields touched by this thesis. For the map compression problem, although we can achieve significant map compression ratio through offline post-processing, the real world is dynamic. If the environment changed, the map needs to be updated and the updated map parts need to be stored in a compatible format to the existing compressed map. If an implicit map format is used (e.g. a foundation model), handling the map compression and update simultaneously can be challenging. For the large-scale NeRF, industrial companies have constructed large simulation environments with the abundant data collected by vehicle fleets. Digesting the large amount of new data into the NeRF system efficiently, while handling the real-world changes at the same time, is also still very challenging.

Bibliography

[1] bzip2. In *http://www.bzip.org/*. 2.1.2

[2] Long-Term Visual Localization Challenges in ICCV 2021. `https://sites.google.com/view/ltvl2021/challenges`, 2021. Accessed: 2022-03-09. 2.1.3

[3] Michal Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager. Vision-Only Robot Navigation in a Neural Radiance World. In *arXiv*, 2021. 6, 6.1, 6.4

[4] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc Van Gool. Soft-to-Hard Vector Quantization for End-to-end Learning Compressible Representations. In *Conf. Neural Inform. Process. Syst.*, 2017. 2.2.1

[5] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. PointnetLK: Robust & Efficient Point Cloud Registration Using PointNet. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 2.1.2, 4.1, 4.2.2

[6] Hernan Badino, Daniel Huber, and Takeo Kanade. The CMU Visual Localization Data Set. `http://3dvis.ri.cmu.edu/data-sets/localization`, 2011. 5.3

[7] Xuyang Bai, Zixin Luo, Lei Zhou, Hongbo Fu, Long Quan, and Chiew Lan Tai. D3Feat: Joint Learning of Dense Detection and Description of 3D Local Features. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020. 2.1.2, 4.1, 4.1, 4.2.3, 4.2.5, 4.3.3

[8] Xuyang Bai, Zeyu Hu, Xinge Zhu, Qingqiu Huang, Yilun Chen, Hongbo Fu, and Chiew-Lan Tai. TransFusion: Robust LiDAR-Camera Fusion for 3D Object Detection with Transformers. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2.1.2

[9] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. In *Int. Conf. Comput. Vis.*, 2021. 2.3, 6.1

[10] Paul J. Besl and Neil D. McKay. A Method for Registration of 3-D Shapes. In

IEEE Trans. Pattern Anal. Mach. Intell., 1992. 2.1.2, 4.1

[11] Peter Biber. The Normal Distributions Transform: A New Approach to Laser Scan Matching. In *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 2003. 2.1.2, 4.1

[12] Samarth Brahmbhatt, Jinwei Gu, Kihwan Kim, James Hays, and Jan Kautz. Geometry-Aware Learning of Maps for Camera Localization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 1.4, 6.1, 6.4.2

[13] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. In *IEEE Trans. on Robotics*, 2016. 5.4

[14] Federico Camposeco, Andrea Cohen, Marc Pollefeys, and Torsten Sattler. Hybrid Scene Compression for Visual Localization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 2.2.2

[15] Chao Cao, Marius Preda, and Titus Zaharia. 3D Point Cloud Compression: A Survey. In *ACM Int. Conf. on 3D Web Technology*, 2019. 2.1.2, 4.1

[16] Song Cao and Noah Snavely. Minimal Scene Descriptions from Structure from Motion Models. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2014. 2.2.2

[17] Nicholas Carlevaris-Bianco, Arash K. Ushani, and Ryan M. Eustice. University of Michigan North Campus Long-Term Vision and Lidar Dataset. In *Int. J. of Robotics Res.*, 2016. 1.4, 6.4.3

[18] Alexandra Carlson, Manikandasriram Srinivasan Ramanagopal, Nathan Tseng, Matthew Johnson-Roberson, Ram Vasudevan, and Katherine A. Skinner. CLONeR: Camera-Lidar Fusion for Occupancy Grid-aided Neural Representations. In *IEEE Robotics and Automation Letters*, 2023. 2.3.1, 6.1

[19] Tim Caselitz, Bastian Steder, Michael Ruhnke, and Wolfram Burgard. Monocular Camera Localization in 3D LiDAR Maps. In *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 2016. 1.1, 2.1.1, 3.3.1

[20] D Cattaneo, M Vaghi, A L Ballardini, S Fontana, D G Sorrenti, and W Burgard. CMRNet: Camera to LiDAR-Map Registration. In *IEEE Int. Conf. Intell. Transportation Syst.*, 2019. 2.1.1, 3, 3.1, 3.2.2, 3.2.3, 3.3, 3.3.1, 3.4

[21] D Cattaneo, M Vaghi, S Fontana, A L Ballardini, and D G Sorrenti. Global Visual Localization in LiDAR-maps through Shared 2D-3D Embedding Space. In *IEEE Int. Conf. Robotics and Automation*, 2020. 2.1

[22] Daniele Cattaneo, Domenico Giorgio Sorrenti, and Abhinav Valada. CMR-Net++: Map and Camera Agnostic Monocular Visual Localization in LiDAR Maps. In *IEEE Int. Conf. Robotics and Automation*, 2020. 1.1, 2.1.1, 3.1, 3.4

[23] Eric R. Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. Pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 2.3.2

[24] Ming-Fang Chang, John W Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. Argoverse: 3D Tracking and Forecasting with Rich Maps. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 1, 1.2, 3.3, 4.2.4

[25] Ming-Fang Chang, Wei Dong, Joshua Mangelson, Michael Kaess, and Simon Lucey. Map Compressibility Assessment for LiDAR Registration. In *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 2021. 1, 1.2, 4, 7

[26] Ming-Fang Chang, Joshua Mangelson, Michael Kaess, and Simon Lucey. HyperMap: Compressed 3D Map for Monocular Camera Registration. In *IEEE Int. Conf. Robotics and Automation*, 2021. 1, 1.1, 2.1.1, 2.2.1, 3, 7

[27] Ming-Fang Chang, Yipu Zhao, Rajvi Shah, Jakob J. Engel, Michael Kaess, and Simon Lucey. Long-term Visual Map Sparsification with Heterogeneous GNN. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 1, 1.3, 2.1.3, 2.1.3, 2.2.2, 5, 7

[28] Ming-Fang Chang, Akash Sharma, Michael Kaess, and Simon Lucey. Neural Radiance Fields with LiDAR Maps. In *Int. Conf. Comput. Vis.*, 2023. 2.3.2, 7

[29] Kuangyi Chen, Huai Yu, Wen Yang, Lei Yu, Sebastian Scherer, and Gui-Song Xia. I2D-Loc: Camera Localization via Image to LiDAR Depth Flow. In *ISPRS J. of Photogrammetry and Remote Sensing*, 2022. 2.1.1

[30] Yu Chen and Guan Wang. EnforceNet: Monocular Camera Localization in Large Scale Indoor Sparse LiDAR Point Cloud. In *arXiv*, 2019. 2.1.1, 3.1

[31] Christopher Choy, Jaesik Park, and Vladlen Koltun. Fully convolutional geometric features. In *Int. Conf. Comput. Vis.*, 2019. 2.1.2, 3.1, 3.2.1, 3.4, 4.1, 4.2.3, 4.3.3

[32] Christopher Choy, Wei Dong, and Vladlen Koltun. Deep Global Registration. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020. 2.1.2, 3.1, 3.2.1, 4.1, 4.2.2

[33] Jiadi Cui and Sören Schwertfeger. CP+: Camera Poses Augmentation with Large-scale LiDAR Maps. In *IEEE Int. Conf. on Real-time Comput. and Robotics*, 2022. 2.1.1

[34] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: Fewer Views and Faster Training for Free. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2.3.1, 6.1, 6.3.3

[35] Yu Deng, Jiaolong Yang, Jianfeng Xiang, and Xin Tong. GRAM: Generative

Radiance Manifolds for 3D-Aware Image Generation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2.3.2

[36] Prafulla Dhariwal and Alex Nichol. Diffusion Models Beat GANs on Image Synthesis. In *Conf. Neural Inform. Process. Syst.*, 2021. 2.3.2

[37] Anh-Dzung Doan, Daniyar Turmukhambetov, Yasir Latif, Tat-Jun Chin, and Soohyun Bae. Learning to Predict Repeatability of Interest Points. In *IEEE Int. Conf. Robotics and Automation*, 2021. 2.1.3, 5.1

[38] Wei Dong, Kwonyoung Ryu, Michel Kaess, and Jaesik Park. Revisiting LiDAR Registration and Reconstruction: A Range Image Perspective. In *arXiv*, 2022. 2.1.2

[39] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An Open Urban Driving Simulator. In *Conf. on Robot Learning*, 2017. 3.3

[40] Renaud Dubé, Andrei Cramariuc, Daniel Dugas, Hannes Sommer, Marcin Dymczyk, Juan Nieto, Roland Siegwart, and Cesar Cadena. SegMap: Segment-based Mapping and Localization Using Data-driven Descriptors. In *Robotics Sci. and Syst.*, 2018. 2.1.2

[41] Marcin Dymczyk, Simon Lynen, Michael Bosse, and Roland Siegwart. Keep It Brief: Scalable Creation of Compressed Localization Maps. In *IEEE/RSJ Int. Conf. Intell. Robots and Syst.* 2.2.2, 5.1

[42] Marcin Dymczyk, Elena Stumm, Juan Nieto, Roland Siegwart, and Igor Gilitschenski. Will It Last? Learning Stable Features for Long-term Visual Localization. In *IEEE Int. Conf. on 3D Vis.*, 2016. 2.1.3, 5.1

[43] Benjamin Eckart, Kihwan Kim, and Jan Kautz. HGMR: Hierarchical gaussian mixtures for adaptive 3D registration. In *Eur. Conf. Comput. Vis.*, 2018. 2.1.2, 4.1, 4.3.2

[44] Gil Elbaz, Tamar Avraham, and Anath Fischer. 3D Point Cloud Registration for Localization Using a Deep Neural Network Auto-encoder. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017. 4.1, 4.2.2, 5

[45] Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles Qi, Yin Zhou, Zoey Yang, Aurelien Chouard, Pei Sun, Jiquan Ngiam, Vijay Vasudevan, Alexander McCauley, Jonathon Shlens, and Dragomir Anguelov. Large Scale Interactive Motion Forecasting for Autonomous Driving : The Waymo Open Motion Dataset. In *Int. Conf. Comput. Vis.*, 2021. 6.1

[46] Kai Fischer, Martin Simon, Florian Ölsner, Stefan Milz, Horst-Michael Groß, and Patrick Mäder. Stickypillars: Robust and efficient feature matching on

point clouds using graph neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 2.1.2

[47] Guy Gafni, Justus Thies, Michael Zollhöfer, and Matthias Nießner. Dynamic Neural Radiance Fields for Monocular 4D Facial Avatar Reconstruction. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 2.3, 6.1

[48] Kyle Gao, Yina Gao, Hongjie He, Dening Lu, Linlin Xu, and Jonathan Li. NeRF: Neural Radiance Field in 3D Vision, A Comprehensive Review. In *arXiv*, 2022. 6

[49] Wei Gao and Russ Tedrake. FilterReg: Robust And Efficient Probabilistic Point-set Registration Using Gaussian Filter and Twist Parameterization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 2.1.2, 4.1, 3, 4.3.2

[50] Abel Gawel, Carlo Del Don, Roland Siegwart, Juan Nieto, and Cesar Cadena. X-View : Graph-Based Semantic Multi-View Localization. In *IEEE Robotics and Automation Letters*, 2018. 2.1

[51] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2012. 1.2, 3.3, 4.2.4, 6.1

[52] Christopher Goodin, Daniel Carruth, Matthew Doude, and Christopher Hudson. Predicting the Influence of Rain on LIDAR in ADAS. In *Electronics*, 2019. 6.3.1, 6.3.4

[53] Albert Gordo, Jon Almazan, Jerome Revaud, and Diane Larlus. Deep Image Retrieval: Learning Global Representations for Image Search. In *Eur. Conf. Comput. Vis.*, 2016. 2.1.3

[54] Cameron Gordon, Shin-Fang Chng, Lachlan MacDonald, and Simon Lucey. On Quantizing Implicit Neural Representations. In *IEEE Winter Conf. Applications of Comput. Vis.*, 2023. 2.2.1

[55] Benjamin Graham and Laurens van der Maaten. Submanifold Sparse Convolutional Networks. In *arXiv*, 2017. 3.1, 3.2.1

[56] M. Greenspan and M. Yurick. Approximate k-d Tree Search for Efficient ICP. In *Int. Conf. 3-D Digital Imaging and Modeling*, 2003. 2.1.2

[57] Gurobi Optimization, LLC. Gurobi - The Fastest Solver. https://www.gurobi.com/, 2021. 5.3

[58] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *Conf. Neural Inform. Process. Syst.*, 2017. 2.1.3, 5.2.2, 5.3.1

[59] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. GANcraft: Unsupervised 3D Neural Rendering of Minecraft Worlds. In *Int. Conf. Comput.*

Vis., 2021. 2.3.1, 2.3.2, 6.1, 6.2.2, 6.2.2

[60] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for Object Segmentation and Fine-grained Localization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2015. 3.1, 3.2.1

[61] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016. 6.2.2

[62] Xun Huang and Serge Belongie. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization. In *Int. Conf. Comput. Vis.*, 2017. 6.4.3

[63] Martin Humenberger, Yohann Cabon, Nicolas Guerin, Julien Morat, Jérôme Revaud, Philippe Rerole, Noé Pion, Cesar de Souza, Vincent Leroy, and Gabriela Csurka. Robust Image Retrieval-Based Visual Localization Using Kapture. In *arXiv*, 2020. 2.1.3, 2.1.3, 5.3, 5.3

[64] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised Learning of Shape and Pose with Differentiable Point Clouds. In *Conf. Neural Inform. Process. Syst.*, 2018. 2.1.1

[65] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017. 1.4, 2.3.2, 6.1, 6.2.2

[66] Alex Kendall, Matthew Grimes, and Roberto Cipolla. PoseNet: A Convolutional Network for Real-Time 6-DoF Camera Relocalization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2015. 2.1

[67] Hyojin Kim, Carlos D. Correa, and Nelson Max. Automatic Registration of LiDAR and Optical Imagery Using Depth Map Stereo. In *Int. Conf. Comput. Photography*, 2014. 1.1, 2.1.1

[68] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Int. Conf. Learn. Represent.*, 2017. 5.2.2, 5.2.3, 5.3.1

[69] Rainer Kümmerle, Michael Ruhnke, Bastian Steder, Cyrill Stachniss, and Wolfram Burgard. Autonomous Robot Navigation in Highly Populated Pedestrian Zones. In *J. of Field Robotics*, 2015. 4.2.4

[70] Orest Kupyn, Volodymyr Budzan, Mykola Mykhailych, Dmytro Mishkin, and Jiri Matas. DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 2.3.2

[71] Sampo Kuutti, Saber Fallah, Konstantinos Katsaros, Mehrdad Dianati, Francis Mccullough, and Alexandros Mouzakitis. A Survey of the State-of-the-Art Localization Techniques and Their Potentials for Autonomous Vehicle Applications.

In *IEEE Internet of Things J.*, 2018. 3.1

[72] Jeong-gi Kwak, Yuanming Li, Dongsik Yoon, Donghyeon Kim, David Han, and Hanseok Ko. Injecting 3D Perception of Controllable NeRF-GAN into StyleGAN for Editable Portrait Image Synthesis. In *Eur. Conf. Comput. Vis.*, 2022. 2.3.2

[73] Xin Lai, Yukang Chen, Fanbin Lu, Jianhui Liu, and Jiaya Jia. Spherical Transformer for LiDAR-based 3D Recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2023. 2.1.2

[74] Alex Junho Lee, Seungwon Song, Hyungtae Lim, Woojoo Lee, and Hyun Myung. (LC)$\hat{2}$: LiDAR-Camera Loop Constraints for Cross-Modal Place Recognition. In *IEEE Robotics and Automation Letters*, 2023. 2.1.1

[75] Yandong Li, Yu Cheng, Zhe Gan, Licheng Yu, Liqiang Wang, and Jingjing Liu. BachGAN: High-Resolution Image Synthesis from Salient Object Layout. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020. 2.3.2

[76] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural Scene Flow Fields for Space-Time View Synthesis of Dynamic Scenes. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 2.3, 2.3.1, 6.1

[77] Zimo Li, Prakruti C. Gogia, and Michael Kaess. Dense Surface Reconstruction from Monocular Vision and LiDAR. In *IEEE Int. Conf. Robotics and Automation*, 2019. 1

[78] Chen-Hsuan Lin, Chen Kong, and Simon Lucey. Learning Efficient Point Cloud Generation for Dense 3D Object Reconstruction. In *AAAI Conf. on Artificial Intell.*, 2018. 2.1.1

[79] Chieh Hubert Lin, Hsin-Ying Lee, Willi Menapace, Menglei Chai, Aliaksandr Siarohin, Ming-Hsuan Yang, and Sergey Tulyakov. InfiniCity: Infinite-Scale City Synthesis. In *arXiv*, 2023. 2.3.2

[80] Tianxiang Lin, Akshay Hinduja, Mohamad Qadri, and Michael Kaess. Conditional GANs for Sonar Image Filtering with Applications to Underwater Occupancy Mapping. In *IEEE Int. Conf. Robotics and Automation*, 2023. 2.3.2, 6.2.2

[81] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural Sparse Voxel Fields. In *Conf. Neural Inform. Process. Syst.*, 2020. 2.3.1, 6.1

[82] Yan Lu, Jiawei Huang, Yi Ting Chen, and Bernd Heisele. Monocular Localization in Urban Environments Using Road Markings. In *IEEE Intell. Vehicles Symposium)*, 2017. 2.1.1

[83] Stefan Luthardt, Volker Willert, and Jürgen Adamy. LLama-SLAM: Learning

High-Quality Visual Landmarks for Long-Term Mapping and Localization. In *IEEE Int. Conf. Intell. Transportation Syst.*, 2018. 2.2.2

[84] Simon Lynen, Bernhard Zeisl, Dror Aiger, Michael Bosse, Joel Hesch, Marc Pollefeys, Roland Siegwart, and Torsten Sattler. Large-Scale, Real-Time Visual-Inertial Localization Revisited. In *Int. J. of Robotics Res.*, 2020. 2.2.2, 5.1, 5.2.4, 5.3, 5.3.1

[85] Dominic Maggio, Marcus Abate, Jingnan Shi, Courtney Mario, and Luca Carlone. Loc-NeRF: Monte Carlo Localization Using Neural Radiance Fields. In *arXiv*, 2022. 6, 6.1

[86] Petri Manninen, Heikki Hyyti, Ville Kyrki, Jyri Maanpää, Josef Taher, and Juha Hyyppä. Towards High-Definition Maps: a Framework Leveraging Semantic Segmentation to Improve NDT Map Compression and Descriptivity. In *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 2022. 2.1.2

[87] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 2.3, 6.1

[88] Julieta Martinez, Sasha Doubov, Jack Fan, Ioan Andrei Bârsan, Shenlong Wang, Gellért Máttyus, and Raquel Urtasun. Pit30M: A Benchmark for Global Localization in The Age of Self-Driving Cars. In *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 2020. 4.1, 4.2.3

[89] Andrew Mastin, Jeremy Kepner, and John Fisher. Automatic Registration of LIDAR and Optical Images of Urban Scenes. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2009. 1.1, 2.1.1

[90] Quan Meng, Anpei Chen, Haimin Luo, Minye Wu, Hao Su, Lan Xu, Xuming He, and Jingyi Yu. GNeRF: GAN-based Neural Radiance Field without Posed Camera. In *Int. Conf. Comput. Vis.*, 2021. 2.3.2

[91] Marcela Mera-Trujillo, Benjamin Smith, and Victor Fragoso. Efficient Scene Compression for Visual-based Localization. In *IEEE Int. Conf. on 3D Vis.*, 2020. 2.2.2, 5.1

[92] Jinyu Miao, Kun Jiang, Yunlong Wang, Tuopu Wen, Zhongyang Xiao, Zheng Fu, Mengmeng Yang, Maolin Liu, and Diange Yang. Poses as Queries: Image-to-LiDAR Map Localization with Transformers. In *arXiv*, 2023. 2.1.1

[93] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Eur. Conf. Comput. Vis.*, 2020. 2.3, 2.3.1, 6, 6.1, 6.2.1, 6.3.2

[94] Arthur Moreau, Nathan Piasco, Dzmitry Tsishkou, Bogdan Stanciulescu, and Arnaud de La Fortelle. LENS: Localization Enhanced by NeRF Synthesis. In *Conf. on Robot Learning*, 2021. 6.1, 6.4.2

[95] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding . In *arXiv*, 2022. 2.3, 6.1

[96] Andriy Myronenko and Xubo Song. Point Set Registration: Coherent Point Drifts. In *IEEE Trans. Pattern Anal. Mach. Intell.*, 2010. 2.1.2, 4.1, 4.2.2

[97] Peter Mühlfellner, Mathias Bürki, M. Bosse, W. Derendarz, Roland Philippsen, and P. Furgale. Summary Maps for Lifelong Visual Localization. In *J. of Field Robotics*, 2016. 2.2.2

[98] Michael Niemeyer and Andreas Geiger. GIRAFFE: Representing Scenes as Compositional Generative Neural Feature Fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 2.3.2

[99] Karen L. Oehler and Robert M. Gray. Combining Image Compression and Classification Using Vector Quantization. In *IEEE Trans. Pattern Anal. Mach. Intell.*, 1995. 2.2.1

[100] Julian Ost, Issam Laradji, Alejandro Newell, Yuval Bahat, and Felix Heide. Neural Point Light Fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2.3.1

[101] Hyun Soo Park, Yu Wang, Eriko Nurvitadhi, James C. Hoe, Yaser Sheikh, and Mei Chen. 3D Point Cloud Reduction Using Mixed-Integer Quadratic Programming. In *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*, 2013. 2.2.2, 5.1, 5.2.4

[102] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable Neural Radiance Fields. In *Int. Conf. Comput. Vis.*, 2021. 2.3, 6.1

[103] Nathan Piasco, Désiré Sidibé, Valérie Gouet-Brunet, and Cedric Demonceaux. Learning Scene Geometry for Visual Localization in Challenging Conditions. In *IEEE Int. Conf. Robotics and Automation*, 2019. 2.1.3

[104] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017. 3.4

[105] Krishna Regmi and Ali Borji. Cross-View Image Synthesis Using Conditional GANs. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 2.3.2

[106] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLP . In *Int.*

Conf. Comput. Vis., 2021. 2.3.1

[107] Konstantinos Rematas, Andrew Liu, Pratul Srinivasan, Jonathan Barron, Andrea Tagliasacchi, Thomas Funkhouser, and Vittorio Ferrari. Urban Radiance Fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2.3.1, 6.1, 6.3.1, 6.3.2, 6.3.3, 6.5

[108] Jerome Revaud, Philippe Weinzaepfel, César De Souza, Noe Pion, Gabriela Csurka, Yohann Cabon, and Martin Humenberger. R2D2: Repeatable and Reliable Detector and Descriptor. In *Conf. Neural Inform. Process. Syst.*, 2019. 2.1.3, 5.3

[109] Barbara Roessle, Jonathan T. Barron, Ben Mildenhall, Pratul P. Srinivasan, and Matthias Nießner. Dense Depth Priors for Neural Radiance Fields from Sparse Input Views. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2.3.1, 6.3.3

[110] Antoni Rosinol, John J. Leonard, and Luca Carlone. NeRF-SLAM: Real-Time Dense Monocular SLAM with Neural Radiance Fields. In *arXiv*, 2022. 6, 6.4

[111] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE Int. Conf. Robotics and Automation*, 2011. 2.1.2

[112] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast Point Feature Histograms (FPFH) for 3D Registration. In *IEEE Int. Conf. Robotics and Automation*, 2009. 2.1.2, 4.1, 4.2.3, 4.3.3

[113] Chitwan Saharia, William Chan, Huiwen Chang, Chris A. Lee, Jonathan Ho, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Palette: Image-to-Image Diffusion Models. In *ACM Trans. Graph.*, 2022. 2.3.2

[114] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Image Super-Resolution via Iterative Refinement. In *IEEE Trans. Pattern Anal. Mach. Intell.*, 2022. 2.3.2

[115] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. SuperGlue: Learning Feature Matching With Graph Neural Networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020. 2.1.2, 2.1.3, 5.1

[116] Paul-Edouard Sarlin, Ajaykumar Unagar, Mans Larsson, Hugo Germain, Carl Toft, Viktor Larsson, Marc Pollefeys, Vincent Lepetit, Lars Hammarstrand, Fredrik Kahl, et al. Back to the Feature: Learning Robust Camera Localization from Pixels to Pose. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021. 2.1.3

[117] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. Fast Image-Based Localization Using Direct 2D-to-3D Matching. In *Int. Conf. Comput. Vis.*, 2011. 2.1

[118] Torsten Sattler, Will Maddern, Carl Toft, Akihiko Torii, Lars Hammarstrand, Erik Stenborg, Daniel Safari, Masatoshi Okutomi, Marc Pollefeys, Josef Sivic,

Fredrik Kahl, and Tomas Pajdla. Benchmarking 6DOF Outdoor Visual Localization in Changing Conditions. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 5.1, 5.3

[119] Torsten Sattler, Qunjie Zhou, Marc Pollefeys, and Laura Leal-Taixe´. Understanding the Limitations of CNN-based Absolute Camera Pose Regression. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 6.4.2

[120] Ruwen Schnabel and Reinhard Klein. Octree-based Point-Cloud Compression. In *Eurographics Symposium on Point-Based Graphics*, 2006. 2.1.2

[121] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-Motion Revisited. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016. 6.1

[122] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise View Selection for Unstructured Multi-View Stereo. In *Eur. Conf. Comput. Vis.*, 2016. 6.1

[123] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis. In *Conf. Neural Inform. Process. Syst.*, 2020. 2.3.2

[124] Aleksandr V Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-ICP. In *Robotics Sci. and Syst.*, 2009. 2.1.2, 4.1

[125] Yan Shen, Zhang Maojun, Lai, Shiming, Liu Yu, and Peng Yang. Image Retrieval for Structure-from-Motion via Graph Convolutional Network. In *Inform. Sci.*, 2021. 2.1.3, 5.1

[126] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. PV-RCNN: Point-voxel Feature Set Abstraction for 3D Object Detection. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020. 4.2.4

[127] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. DeepVoxels: Learning Persistent 3D Feature Embeddings. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 2.1.1

[128] Paloma Sodhi, Michael Kaess, Mustafa Mukadam, and Stuart Anderson. Patch-Graph: In-hand Tactile Tracking with Learned Surface Normals. In *IEEE Int. Conf. Robotics and Automation*, 2022. 2.3.2, 6.2.2

[129] Deqing Sun, Xiaodong Yang, Ming Yu Liu, and Jan Kautz. PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 2.1.1, 3.2.3

[130] Linus Svarm, Olof Enqvist, Fredrik Kahl, and Magnus Oskarsson. City-Scale Localization for Cameras with Known Vertical Direction. In *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017. 2.1

[131] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Milden-

hall, Pratul Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. Block-NeRF: Scalable Large Scene Neural View Synthesis. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2.3.1, 6.1, 6.2.1, 6.3.1, 6.3.2, 6.5

[132] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. In *Int. Conf. Comput. Vis.*, 2019. 2.1.2

[133] Yurun Tian, Xin Yu, Bin Fan, Fuchao Wu, Huub Heijnen, and Vassileios Balntas. SoSnet: Second Order Similarity Regularization for Local Descriptor Learning. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 2.1.3

[134] Carl Toft, Carl Olsson, and Fredrik Kahl. Long-term 3D Localization and Pose from Semantic Labellings. In *Int. Conf. Comput. Vis. Worksh.*, 2017. 2.1.3

[135] Carl Toft, Erik Stenborg, Lars Hammarstrand, Lucas Brynte, Marc Pollefeys, Torsten Sattler, and Fredrik Kahl. Semantic Match Consistency for Long-Term Visual Localization. In *Eur. Conf. Comput. Vis.*, 2018. 2.1.3

[136] Carl Toft, Will Maddern, Akihiko Torii, Lars Hammarstrand, Erik Stenborg, Daniel Safari, Masatoshi Okutomi, Marc Pollefeys, Josef Sivic, Tomas Pajdla, et al. Long-Term Visual Localization Revisited. In *IEEE Trans. Pattern Anal. Mach. Intell.*, 2020. 2.1.3

[137] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017. 2.1.1

[138] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Conf. Neural Inform. Process. Syst.*, 2017. 2.1.3

[139] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *Int. Conf. Learn. Represent.*, 2018. 2.1.3, 5.1, 5.2.2, 5.3.1

[140] Ji Wan, Dayong Wang, Steven Chu Hong Hoi, Pengcheng Wu, Jianke Zhu, Yongdong Zhang, and Jintao Li. Deep Learning for Content-Based Image Retrieval:A Comprehensive Study. In *ACM Int. Conf. Multimedia*, 2014. 2.1.3

[141] Jingkang Wang, Sivabalan Manivasagam, Yun Chen, Ze Yang, Ioan Andrei Bârsan, Joyce Anqi Yang, Wei-Chiu Ma, and Raquel Urtasun. CADSim: Robust and Scalable in-the-wild 3D Reconstruction for Realistic and Controllable Sensor Simulation. In *Conf. on Robot Learning*, 2022. 6.2.1

[142] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep Graph Library: A Graph-Centric, Highly-

Performant Package for Graph Neural Networks. In *arXiv*, 2020. 5.2.2, 5.3, 6.3.2

[143] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 2.3.2

[144] Wei Wang, Jun Liu, Chenjie Wang, Bin Luo, and Cheng Zhang. DV-LOAM: Direct Visual LiDAR Odometry and Mapping. In *Remote Sensing*, 2021. 2.1.2

[145] Weilun Wang, Jianmin Bao, Wengang Zhou, Dongdong Chen, Dong Chen, Lu Yuan, and Houqiang Li. Semantic Image Synthesis via Diffusion Models. In *arXiv*, 2022. 2.3.2

[146] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Peng Cui, P. Yu, and Yanfang Ye. Heterogeneous Graph Attention Network. In *The World Wide Web Conf.*, 2021. 2.1.3

[147] Yue Wang and Justin Solomon. Deep Closest Point: Learning representations for point cloud registration. In *Int. Conf. Comput. Vis.*, 2019. 2.1.2, 4.1, 4.2.2

[148] Xinkai Wei, Ioan Andrei Barsan, Shenlong Wang, Julieta Martinez, and Raquel Urtasun. Learning to Localize through Compressed Binary Maps. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 2.2.1

[149] Yi Wei, Shaohui Liu, Yongming Rao, Wang Zhao, Jiwen Lu, and Jie Zhou. NerfingMVS: Guided Optimization of Neural Radiance Fields for Indoor Multi-view Stereo. In *Int. Conf. Comput. Vis.*, 2021. 6.1

[150] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, Deva Ramanan, Peter Carr, and James Hays. Argoverse 2: Next Generation Datasets for Self-Driving Perception and Forecasting. In *Conf. Neural Inform. Process. Syst.*, 2021. 1.4, 6.1, 6.2.4, 6.3.1

[151] Ryan W. Wolcott and Ryan M. Eustice. Visual Localization within LIDAR Maps for Automated Urban Driving. In *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 2014. 2.1.1

[152] Ryan W. Wolcott and Ryan M. Eustice. Fast LiDAR Localization Using Multiresolution Gaussian Mixture Maps. In *IEEE Int. Conf. Robotics and Automation*, 2015. 2.1.1

[153] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019. 1.4, 6.1, 6.4.1

[154] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli,

and Ulrich Neumann. Point-NeRF: Point-based Neural Radiance Fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2.3.1, 6.1, 6.2.1, 6.2.1, 6.2.1, 6.3.2, 6.5, 6.5

[155] Chen Yang and Gérard Medioni. Object Modelling by Registration of Multiple Range Images. In *IEEE Int. Conf. Robotics and Automation*, 1992. 4.1

[156] Heng Yang, Jingnan Shi, and Luca Carlone. TEASER: Fast and Certifiable Point Cloud Registration. In *IEEE Trans. on Robotics*, 2020. 1.2, 2.1.2, 4.1, 4.3.3, 4.4

[157] Jiaolong Yang, Hongdong Li, Dylan Campbell, and Yunde Jia. Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration. In *IEEE Trans. Pattern Anal. Mach. Intell.*, 2016. 2.1.2, 4.1, 4.2.2, 4.3.1

[158] Zi Jian Yew and Gim Hee Lee. 3DFeat-Net: Weakly Supervised Local 3D Features for Point Cloud Registration. In *Eur. Conf. Comput. Vis.*, 2018. 4.2.5

[159] Huan Yin, Yue Wang, Li Tang, Xiaqing Ding, Shoudong Huang, and Rong Xiong. 3D LiDAR Map Compression for Efficient Localization on Resource Constrained Vehicles. In *IEEE Int. Conf. Intell. Transportation Syst.*, 2021. 2.1.2, 4.1, 4.4

[160] Huan Yin, Xuecheng Xu, Sha Lu, Xieyuanli Chen, Rong Xiong, Shaojie Shen, Cyrill Stachniss, and Yue Wang. A Survey on Global LiDAR Localization: Challenges, Advances and Open Problems. In *arXiv*, 2023. 2.1.2

[161] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, and Qinhong Chen. Plenoxels: Radiance Fields without Neural Networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2.3.1

[162] Huai Yu, Weikun Zhen, Wen Yang, Ji Zhang, and Sebastian Scherer. Monocular Camera Localization in Prior LiDAR Maps with 2D-3D Line Correspondences. In *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 2020. 2.1.1

[163] Wentao Yuan, Benjamin Eckart, Kihwan Kim, Varun Jampani, Dieter Fox, and Jan Kautz. DeepGMR: Learning Latent Gaussian Mixture Models for Registration. In *Eur. Conf. Comput. Vis.*, 2020. 2.1.2, 4.1, 4.2.2, 4

[164] Huangying Zhan, Jiyang Zheng, Yi Xu, Ian Reid, and Hamid Rezatofighi. ActiveRMAP: Radiance Field for Active Mapping And Planning. In *arXiv*, 2022. 6, 6.1

[165] Ce Zhang, Chengjie Zhang, Yiluan Guo, Lingji Chen, and Michael Happold. MotionTrack: End-to-End Transformer-based Multi-Object Tracking with LiDAR-Camera Fusion. In *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*, 2023. 2.1.2, 2.2.2

[166] Ji Zhang and Sanjiv Singh. LOAM: Lidar Odometry and Mapping in Real-time.

In *Robotics Sci. and Syst.*, 2014. 1.1

[167] Qian Yi Zhou, Jaesik Park, and Vladlen Koltun. *Open3D: A modern library for 3D data processing.* arXiv, 2018. 2.1.2, 3.1, 3.2.1

[168] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *Int. Conf. Comput. Vis.*, 2017. 2.3.2, 6.2.2, 6.2.2

[169] Zheng Zou, Hong Lang, Yuexin Lou, and Jian Lu. Plane-based Global Registration for Pavement 3D Reconstruction Using Hybrid Solid-state LiDAR Point Cloud. In *Automation in Construction*, 2023. 2.1.2